

Defense Information Infrastructure (DII)
Common Operating Environment (COE)

Windows NT 4.0
Programmer's Manual

Version 3.1

2 April 1997

Prepared for:
Defense Information Systems Agency

Table of Contents

Foreword	v
1. Introduction	1
1.1 Purpose	1
1.2 Background	1
1.3 Scope	2
1.4 Document Structure	2
2. DII COE Concept Overview	5
2.1 The DII COE Foundation	5
2.2 COE Concept	7
2.3 Model & Principles	7
2.4 Design and Development Considerations	9
2.4.1 System Concepts for Programming	9
2.4.2 Building for Integration	10
2.4.2.1 Rightsizing of Applications	10
2.4.2.2 Application/Data Separation	10
2.4.2.3 Design for Reuse and Porting	11
2.4.3 Compliance	11
2.4.4 Environment Stability	12
2.4.5 Division of Computing Tasks	12
2.4.6 OSE Enabled COE Applications	13
2.4.7 Distributed COE Applications	14
2.5 Integration and Runtime Specification	14
2.5.1 Runtime Environment	15
2.5.1.1 Segment Types	15
2.5.1.2 Segment Prefixes and Reserved Symbols	16
2.5.1.3 Disk Directory Layout	16
2.5.2 Variants	16
2.5.3 System Management	17
2.5.4 Development Process	17
2.5.4.1 The COE Development Process	17
2.5.4.1.1 Segment Registration	18
2.5.4.1.2 Segment Development	18
2.5.4.1.3 Segment Submission	19
2.5.4.1.4 Segment Integration	19
2.5.4.1.5 Segment Installation	20
2.5.4.2 Migration Considerations	20
2.5.4.3 Development Environment	21
2.5.4.3.1 Development Scripts	21
2.5.4.3.2 Directory Structures	21
2.5.4.3.3 Private and Public Files	21

Table of Contents (continued)

2.5.4.3.4	Developer's Toolkit	21
2.5.5	Database Considerations	22
2.5.5.1	Database Segmentation Principles	22
2.5.5.1.1	Database Segments	22
2.5.5.1.2	Database Segmentation Responsibilities	22
2.5.5.2	RDBMS Tuning and Customization	22
2.5.5.3	Database Inter-Segment Dependencies	23
3.	DII COE Kernel	25
3.1	Operating System	25
3.2	Windowing	26
3.3	Desktop	26
3.3.1	Desktop Tools	26
3.3.2	Common Look and Feel	27
3.4	System Administration	28
3.4.1	COE Runtime Tools	28
3.4.2	Print Services	29
3.4.2.1	Remote Queue Administration	29
3.4.2.2	Printer Configuration/Administration	29
3.4.2.3	Support for Remote LAN Printing	29
3.4.2.4	Print Service Segments	30
3.4.3	Process Management	30
3.4.4	Session Management	30
3.5	Security Administration	30
3.5.1	Account Groups	30
3.5.2	Security Services	31
3.5.3	Security Software	32
3.6	Distributed Computing Environment	32
3.6.1	Distributed Computing Environment Cells	33
3.6.2	Distributed Computing Environment Services	33
3.6.3	Distributed Computing Environment Supplemental Services	34
4.	DII COE Developer Toolkit and API's	35
4.1	COE Tools	35
4.2	Application Program Interface	35
4.2.1	Concept	35
4.2.2	Role of APIs in Application Development	36
4.3	Standards	37
5.	DII COE Infrastructure Services	39
5.1	Management Services	39
5.1.1	System Management Services	39
5.1.1.1	Management Environment	39

Table of Contents (continued)

5.1.1.2	Administration	40
5.1.1.3	Software Distribution	41
5.1.1.4	Monitoring	41
5.1.2	Network Management Services	41
5.1.2.1	Management Information Base (MIB) Functionality	41
5.1.2.1.1	Host Resources MIB	41
5.1.2.1.2	Network Management MIB	42
5.1.2.1.3	Remote Network Management MIB	43
5.1.2.2	COE Functionality	43
5.1.3	Print Management Services	44
5.2	Communications Services	44
5.2.1	COE Communications Software / USA Comm Server	44
5.2.2	Unified Build (UB) Comms Service	44
5.3	Data Management Services	44
5.3.1	SHADE Concepts	44
5.3.2	Database Management Systems	45
5.3.3	Developer Constraints	45
5.3.4	Database Integration	45
5.3.5	Creating Database Objects	47
5.3.6	Inter-Database Dependencies	48
5.4	Network Services	48
5.4.1	Distributed Computing Environment	48
5.4.1.1	Cell Management	49
5.4.1.2	Distributed File Service	49
5.4.2	Internet Services	49
5.5	Utilities	49
5.5.1	General Utilities	49
5.5.1.1	File Related Utilities	50
5.5.1.2	UNIX Script Utilities	50
5.5.1.3	Windows Emulation	50
5.5.2	Security Utilities	50
5.5.2.1	Probes	50
5.5.2.2	Sentries	51
6.	DII COE Support Applications	53
6.1	Office Automation	53
6.2	Mapping, Charting, Geodesy, & Imagery	53
6.3	Messaging	57
6.3.1	User Services	57
6.3.2	Application Support	57
6.4	Alerts	58
6.5	Correlation	59
6.6	Situation Display	59

Table of Contents (continued)

7.	DII COE User Information	61
7.1	POSIX Calls	61
7.2	Reference Documentation	61
7.3	Extending the COE	61
7.4	Problem Reporting	62
7.5	COE On-line Services	62
7.5.1	Security Features	63
7.5.2	COE Information Server (CINFO)	63
Appendix A: Acronym List		65
Appendix B: Glossary		69
Appendix C: Reference Documents		73
Appendix D: List of JTA Mandated Standards		75
Appendix E: Programming Guide for Windows NT 4.0		79

List of Tables

Table C-1.	List of Referenced Documents	73
Table C-2.	COE Documentation Cross-Reference	74
Table E-1.	Segment Descriptor Files	87
Table E-2.	SegInfo Descriptor Sections	88

List of Figures

Figure 2-1.	DII COE and COE Based Systems	6
Figure 2-2.	DII COE Model	8
Figure 2-3.	Division of Client/Server Tasks	13
Figure 3-1.	Distributed Computing Environment Services	33
Figure 5-1.	Business Rules and Constraints	48
Figure E-1.	Segment Directory Structure	84

Foreword

Details regarding the Defense Information Infrastructure (DII) Common Operating Environment (COE) Commercial-Off-The-Shelf (COTS) licensed products do not appear in this release of the DII COE Programmer's Manual. Contact the Defense Information Systems Agency (DISA) for information concerning the licensed software and supporting documentation.

NOTE: Contact DISA's Configuration Management for the available documentation on the DII COE COTS licensed applications.

The DII COE Programmer's Manual is part of a set of DII COE developer documentation published in conjunction with major or general releases of the DII COE (such as DII COE Version 3.0). It contains the latest information available on the date of release of this publication.

Segment Availability: Not all DII COE segments described in this document may be available in the referenced release of the DII COE. The information contained herein may precede the availability of certain DII COE software segments. Please visit the DISA's DII COE Home Page on the World Wide Web to obtain the most current information available and to obtain the latest available release of DII COE software segments and other related documentation required by your organization.

DISA DII COE Home Page URL: <http://www.disa.mil/dii/diicoe> or
<http://204.34.175.79/dii/>

Segment Documentation Applicability: Software segments included in this document are identified by their DII COE segment release version number and/or product version number, as available. It is important to note, however, that documentation released with a given segment version may be applicable to subsequent version(s) of the same software segment. For example, the installation guide for a DII segment version 3.0.0.3 may also apply to version 3.0.0.4 of the same segment unless superseded by a new release of the documentation. Refer to the details provided in the Version Description Document (VDD) for a particular segment release and its related amendments or errata to obtain the most current information on the fixes incorporated, additional sources of information, or reference documents.

This page intentionally left blank.

1. Introduction

1.1 Purpose

The *Defense Information Infrastructure (DII) Common Operating Environment (COE) Windows NT 4.0 Programmer's Manual* (referred to in this document as the DII COE Programmer's Manual) is a broad-based description of the DII COE to effectively communicate how the available services are employed by the developers of mission support applications. This manual has been created, in conjunction with the DII COE Programmer's Reference Manual (referred to hereafter as the DII COE Programmer's Reference Manual), to provide COE users and programmers with a manual for system and software development within the DII COE environment.

This manual provides an overview of the COE design and development concepts and the requirements and standards that apply in this environment, describes the services currently available through the COE, and provides a general "how to use" discussion for these services.

The Defense Information Systems Agency (DISA) will review and update this document as required to remain current with the evolution of the DII COE. This document supersedes Version 3.0, all earlier draft versions, presentations, or working group notes. Please direct any comments or questions regarding the DII COE Programmer's Manual to:

Point of Contact (POC)	Cmdr. Charles B. Cameron Chief, DII COE Engineering Division
Address	DISA/JIEO/JEXF-OSF 45335 Vintage Park Plaza Sterling, VA 20166-6701
Telephone	(703) 735-8825
Fax	(703) 735-8761
Internet Address	cameronc@ncr.disa.mil

1.2 Background

The national military strategy dictates that United States (US) forces be able to project power from Continental US (CONUS) bases, sanctuaries, and in-theater locations to an area of conflict. In addition, US forces must support humanitarian missions, often on short notice, anywhere in the world. These military missions make the Services increasingly reliant on long-distance communications and information services of all kinds. The DII is intended to provide the Military Services (and all other Department of Defense (DoD) elements) with information management capabilities to meet these and other DoD missions. The DII COE is the mechanism used to build and integrate the systems and services of the DII.

The purpose and vision of the DII are detailed in the 28 July 1995, DII Master Plan, Version 3.0, which summarizes the primary guidance to the DII integration effort. The DII is defined as a seamless web of communication networks, computers, databases, applications, and the associated people and facilities that meet the information processing and transport needs of DoD users in peace and all crisis, conflict, humanitarian support, and wartime roles. It includes: (1) the facilities to transmit, store, process, and display voice, data, and images; (2) the data, such as video programming, databases, and other media; (3) the applications and software; (4) the network standards and protocols; and (5) the resources to design, develop, construct, manage, and operate the DII.

The DII is a departmental asset. It is the sum of all parts that constitute the information management assets owned by each of the components, including the Office of the Secretary of Defense (OSD), the Joint Chiefs of Staff, the individual Services, Agencies, and others. The Principal Staff Assistants (PSAs) (including the Joint Staff), the Commanders in Chief (CINC), Services, and Agencies (C/S/As), and DISA share the responsibility for policies, budgets, installations, and operations of the DII. PSAs are responsible for planning and funding functional applications while DISA is responsible for the implementation of the support infrastructure. These applications depend on the shared infrastructure provided by the DII COE.

1.3 Scope

This document describes the concepts, models and architecture of the DII COE, its major components, and the resources provided for COE development activities. It also describes the requirements for building and integrating software components on top of the DII COE. This document provides implementation concepts that describe the following, from the perspective of DII development:

- C DII COE concept, model, and principles,
- C Major components of the DII COE,
- C Resources provided for COE development activities,
- C Requirements for COE software compliance and registration, and
- C DII COE runtime environment.

1.4 Document Structure

This DII COE Programmer's Manual is divided into seven chapters. Each chapter is devoted to a topic of primary importance in DII COE development activity.

Chapter 1 discusses the purpose, background, and scope of this document. It also contains an overview of the DII.

Chapter 2 describes the concepts and requirements for the DII COE to support DoD services and mission applications, the COE design and development considerations, and an introduction to the *Defense Information Infrastructure (DII) Common Operating Environment (COE) Integration and Runtime Specification* (referred to as the DII COE I&RTS).

Chapter 3 describes the functionality provided by the DII COE Kernel and introduces the COE environment. It includes an introduction to the Application Program Interface (API) tools available to assist the COE application developers.

Chapter 4 describes the developer's toolkit and Application Program Interfaces (APIs) and identifies the various elements delivered in the DII COE support services.

Chapter 5 describes the support infrastructure services consisting of management services that a developer may use in programming various mission and component software.

Chapter 6 describes support applications that are available in creating DII COE segments and identifies the major applications to essential functions including the desktop office automation.

Chapter 7 provides basic information resources in support of the users of the DII COE including user information for the extension of the COE, security, and on-line services.

Appendices There are 5 appendices (A through E) which contain the following:

- A: Acronym List
- B: Glossary
- C: References
- D: List of Joint Technical Architecture (JTA) Mandated Standards
- E: Programming Guide for Solaris and HP

This page intentionally left blank.

2. DII COE Concept Overview

The DII effectively integrates people and organizations, data and information, and work processes and computing services across the DoD. The DII COE consists of architecture, standards, and reusable software components. The DII COE is a “plug and play” open architecture designed around a client/server model. The DII COE concept is best described as an architecture that is fully compliant with the *Department of Defense Technical Architecture Framework for Information Management* (TAFIM) and meets the requirements of the DoD JTA.

The DII COE concept provides a distributed application infrastructure that promotes DoD-wide interoperability, portability, and scalability. Its objective is to provide access to data, independent of location, in a reliable, cost-efficient manner. The DII COE emphasizes both software reuse and data reuse. Shared Data Environment (SHADE) is the data reuse strategy for the DII COE. Refer to Chapter 1 of the DII COE I&RTS, Version 3.0 dated January 1997, for details about what the COE encompasses.

2.1 The DII COE Foundation

The COE is an Open System Environment (OSE) that provides the ‘foundation’ for building an open system and provides a practical update mechanism for operational sites. Figure 2-1 shows how the DII COE serves as a foundation for building multiple systems. The diagram shows two types of reusable software: the operating system, and the COE components. Chapter 2 of the DII COE I&RTS describes the COE components in more detail. It is sufficient to note that access to these components is through Application Program Interfaces (APIs) and they form the architectural backbone of the target system. Each system built upon the COE foundation uses the same set of APIs to access common COE components, the same approach to integration, and the same set of tools for enforcing COE principles. Precisely the same COE software components are used for common functions, such as communications interfaces and data flow management.

The DII COE is an open architecture that is not tied to a specific hardware platform. It uses Portable Operating System Interface for UNIX (POSIX) -compliant operating systems and industry standards such as X-Window and Motif. The present COE taxonomy and architectural design requirements are detailed in the *Architectural Design Document for the Defense Information Infrastructure (DII) Common Operating Environment (COE)*, dated January 1996.

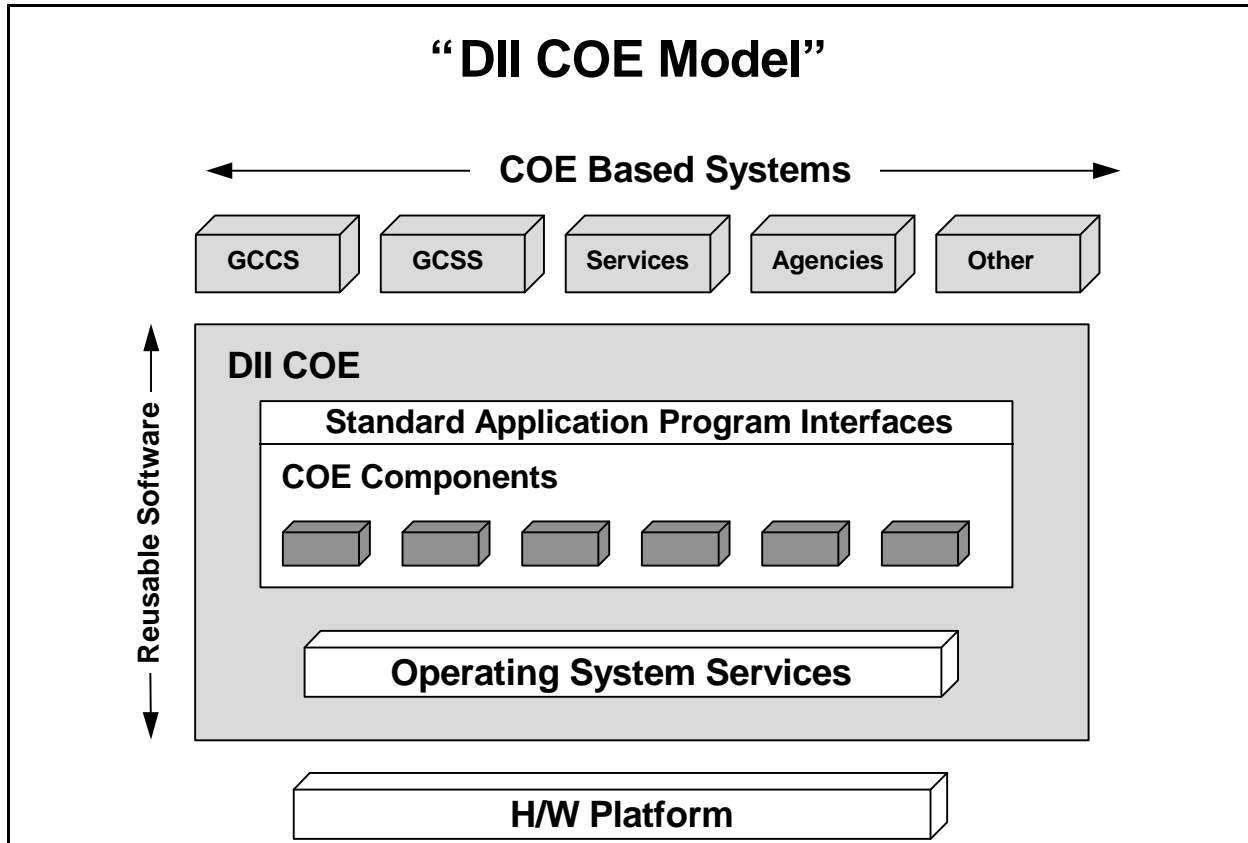


Figure 2-1. DII COE and COE Based Systems

The DII COE architecture is divided into two major areas: Platform Services and Common Support Applications. Platform Services are those types of services that support the flow of information, while Common Support Applications are critical to interoperability but do not directly support the flow of information.

Platform Services include the following functions:

- C Management Services
- C Security Services
- C Communications Services
- C Distributed Computing
- C Data Management Services
- C Presentation Services

Common Support Applications include the following functions:

- C Office Automation
- C Mapping, Charting, Geodesy and Imagery (MCG&I) Service
- C Message Processing Service

- C Alert Services
- C Correlation Service
- C Situation Display Service

The COE is also an evolutionary acquisition and implementation strategy. It emphasizes incremental development and fielding to reduce the time required to put new functionality into the hands of the warrior, while maintaining software quality and minimizing program risk and cost.

2.2 COE Concept

In COE-based systems all software, except the operating system and basic windowing software, is packaged in self-contained units called ‘segments.’ Segments are the most basic building blocks and are defined in terms of the functions they perform. They may contain one or more Computer Software Configuration Items (CSCIs). Reusable segments that are part of the COE are known as COE components or simply ‘components.’ The principles governing how segments are loaded, removed, or interact with one another are the same for all segments. Selecting software modules that fulfill the COE component requirements, and extending them to meet other problem domains, is an ongoing task. The COE preserves backward compatibility so that mission applications are not abandoned just because there is an update of the COE.

The DII COE is structured as a “plug and play” architecture. Migration of existing legacy systems to the COE is conceptually straightforward but may require considerable effort due to the requirement to switch to a different set of building blocks. That is, the effort may not be in conforming to a new architectural concept but in modifying code to use a common set of APIs. The ‘plug and play’ concept clearly conveys the goal and the simplicity that most segment developers will encounter. The DII COE I&RTS Chapter 2 clarifies the basic understanding of the segments and the distinction between interoperability and integration.

2.3 Model & Principles

This subsection introduces the DII COE Model and Structure of the development environment. It discusses the basic principles and structure of the DII COE. Additional details about the DII COE tools and APIs can be found in the DII COE Programmer’s Reference Manual.

Figure 2-2 illustrates the relationship between the COE, component segments, and mission application segments. The COE encompasses Government-Off-The-Shelf (GOTS) software, Commercial-Off-The-Shelf software (COTS), the operating system, windowing software, tools, and APIs. Refer to the *User Interface Specifications for the Defense Information Infrastructure (DII)* (DII COE Style Guide), Version 2.0, dated 1 April 1996, the DII COE I&RTS, and the Architecture Design Document for the DII COE for more details about COE model and principles. Figure 2-2 is a generic diagram intended only to show relationships and relative position of the services, APIs, and the mission applications. The labeled boxes show services can be useful for the Command, Control, Communications, Computers, and Intelligence (C4I), logistics, and mission application domains while the principles remain unchanged. The diagram

remains applicable even when service boxes are replaced with examples of another problem domain.

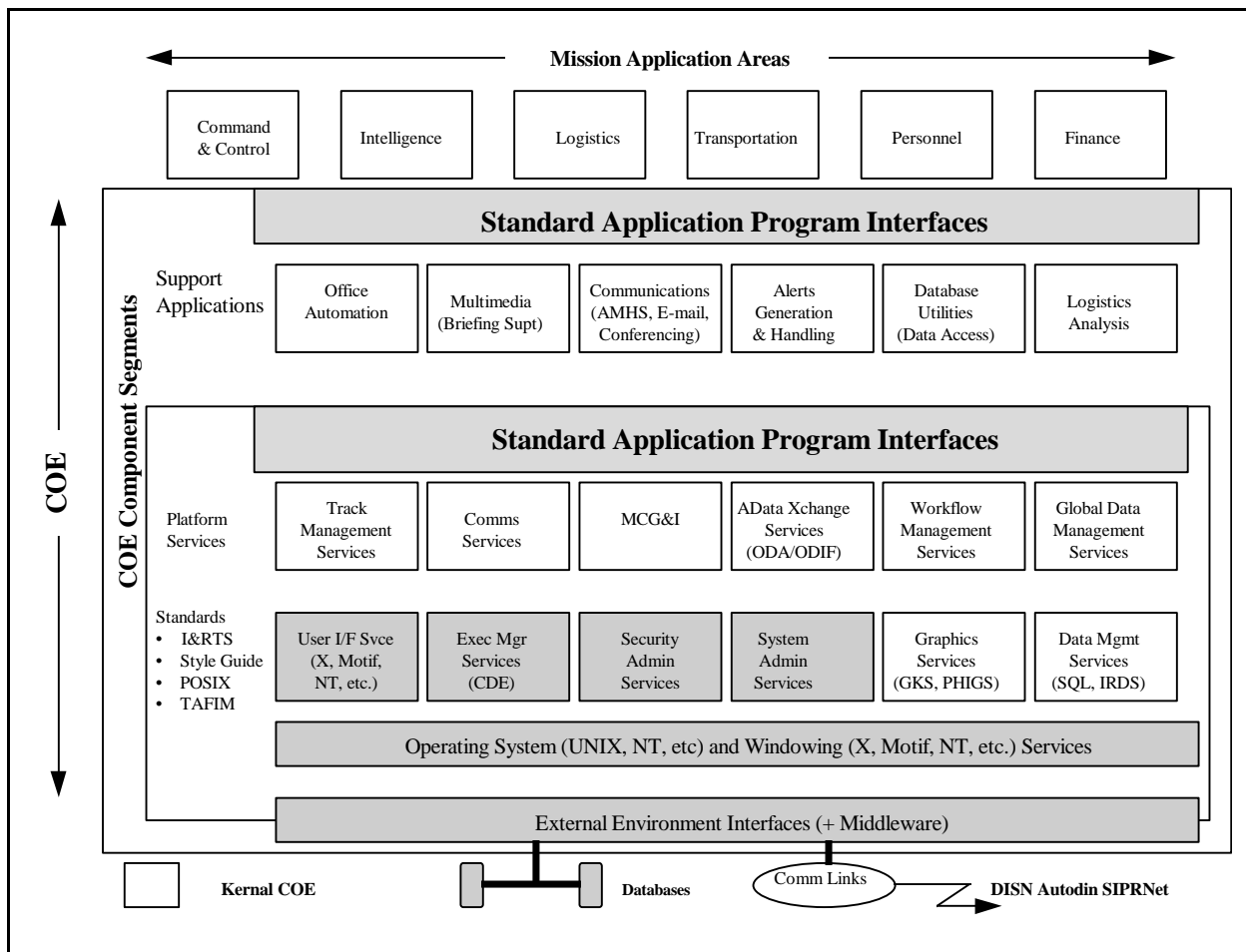


Figure 2-2. DII COE Model

The COE is a collection of building blocks where the operating system, windowing environment, and COE core services form the COE backbone (called the kernel) that other COE component segments “plug” into. Some of the COE components are required, such as the kernel components, while other components are optional depending upon the system being built. Selection of the specific software modules which comprise the COE determines which problem domain(s) can be addressed by the particular COE installation. Chapter 2 of the DII COE I&RTS details the determination of what functions the COE is required to perform and the set of criteria for selecting the software components which perform the required functions.

The DII COE APIs define how segments may interconnect. Figure 2-2 also implies that APIs are the only avenue for accessing support application and platform services provided by the COE. This is true for all COE software, including COTS software. However, the COE does not create an additional layer on top of the COTS software. These components may be accessed directly

using vendor-supplied APIs for these commercial products as long as such usage does not circumvent the COE model (i.e., conflict with the COE runtime environment). For example, the COE supports POSIX-compliant operating systems. Some vendors provide non-POSIX compliant extensions to the operating system services. Use of such extensions, even when readily available through vendor supplied APIs, is not allowed since such usage violates the COE architecture and inhibits interoperability, portability, and reuse.

Physical databases (e.g., MIDS, Integrated Database (IDB), Joint Operations Planning and Execution System (JOPES) database, etc.) are not considered part of the COE although the software, such as the Relational Database Management System (RDBMS) which accesses and manages the data, is part of the COE. Within this conceptual model, the RDBMS functions as the COE's disk controller and disk drives. The application's databases, which are not part of the COE, can be equated to directories or partitions on the drives accessed through the RDBMS disk controller. Data objects belonging to each database can be considered as files within those directories.

The precise configuration of COTS products used in the COE is under strict configuration control. This is necessary because configurable items, such as the amount of shared memory or swap space, must be known and carefully controlled in order for other components in the COE to operate properly. For this reason, COTS products are assigned a DII COE version number in addition to the vendor-supplied version number so as to be able to track and manage configuration changes.

A fundamental principle throughout the COE is that segments are not allowed to directly modify any resource "owned" by another segment. This includes files, directories, modifications to the operating system, and modification to windowing environment resources. Instead, the COE provides tools through which a segment can request extensions to its base environment. The importance of this principle cannot be overemphasized because environmental interactions between software components are a primary reason for integration difficulties. By providing software tools which arbitrate requests to extend the environment, integration can be largely automated and potential problem areas can be identified.

2.4 Design and Development Considerations

This subsection describes the essential elements for the design and development of applications using the DII COE tools and APIs. It is not intended to be a comprehensive guide for programming. Rather, it provides the concepts and considerations in the design of client/server-based distributed applications for DII. For more comprehensive design and development for life-cycle maintenance, refer to the Software Engineering Institute (SEI) Capability Maturity Model (CMM).

2.4.1 System Concepts for Programming

Segments provide functionality that is easily added to or removed from the target system in small manageable units. Segments are defined in terms of functions that are meaningful to operators,

not in terms of internal software structure. Structuring the software into segments is a powerful concept that allows considerable flexibility for configuring the system to meet specific mission needs or to minimize hardware requirements for an operational site.

Integration is not possible without strict standards that describe how to properly build components to add to the system. The requirements for DII COE compliant segments are defined in the DII COE I&RTS and the DII COE Style Guide. The DII COE provides automated tools to measure compliance and to pinpoint problem areas. A useful side effect of the tools and procedures is that software integration is largely an automated process, thus significantly reducing development time while automatically detecting potential integration and runtime problem areas.

2.4.2 Building for Integration

Integrated applications are the main focus of the DII mission because of their importance to the DoD efforts in applications interoperability, portability, scalability, and component reuse. Integration of mission applications is paramount to system operations and the user community within the DII. However, system's application components within a platform require a high level of integration to assure component independence of software versions, enhancements and upgrades.

2.4.2.1 Rightsizing of Applications

“Rightsizing” requires the optimum division and allocation of computing tasks between client and server for distributed applications. Rightsizing also takes into consideration the requirements for custom-designed mission applications and the integration of DII COE components. Consideration of the network service requirements is also essential in providing a context for a distributed application design. The division of computing tasks is a design issue in mission applications and may affect the relative stability of the COE components used.

A well-planned integrated application is network infrastructure-dependent and provides an environment that may grow or shrink dynamically or progressively from local to regional to global interactions as the DoD mission requirements change. The progression from simple (local) to universal (global) use incrementally increases the complexity of applications and system deployment, operations, and support. However, the basic distributed applications, as designed, should not change in the process.

2.4.2.2 Application/Data Separation

Achieving a high level of application portability and component reuse requires development methods and tools that guarantee a level of independence. The separation of application processing from the semantics of data is a primary consideration to develop programs that are independent of the vendor-specific database or data warehouse in use. Similarly the DII COE infrastructure (as defined by TAFIM and related OSE standards) requires the independence of application components from operating system services. Separation of those services that are native to the computer in use, versus standards-defined POSIX services and interfaces, is the key.

This separation assures the means to create portable and reusable applications components and common functional modules (or subroutine calls) that interact with mission applications to provide the highest degree of cross-application integration.

2.4.2.3 Design for Reuse and Porting

The DII COE application development services support the reuse and portability of mission applications. These services use concepts, methods and infrastructure defined by the TAFIM and related OSE standards. The COE provides the means to create portable and reusable applications components (and common functional modules) that interact with other applications residing in different computers regardless of the platform.

The most important aspect of developing mission essential portable applications is to scope the tasks and clearly define the objectives of the client and the server. A distributed application that uses OSE application services must use APIs. APIs are primarily standards-based and provide the appropriate programming language bindings (i.e., Ada and C).

Specialized or mission-critical distributed application capability requires an in-depth understanding of the business process and operational requirements. But foremost, the designer must understand the functional limitations of the environment that is used or created.

Understanding the methods and automating the business/operations processes are not sufficient to develop an optimum distributed processing environment. Sometimes, a redesign of the business process is necessary to maximize the benefit of distributed computing. Another aspect of designing and developing a distributed application is understanding the current capabilities of the network services and the planned future enhancements. This is important because applications need to scale and transition to future technologies and new mission realities without excessive constraints. Some considerations and elements are as follows:

- C Understanding the mission and the processes required,
- C Network responsiveness to the enterprise needs,
- C Seamless distributed processing across the network,
- C “Reachability” -- local, area, regional, national, international.

2.4.3 Compliance

The COE compliance requirements ensure “plug and play” integration of segments into the COE by measuring compliancy quantitatively. The four main compliance categories are as follows:

- C Runtime Environment
- C Style Guide
- C Architectural Compatibility
- C Software Quality

Levels of compliance for each of these categories are measured objectively through a set of checklists. Compliance checking is done on a segment-by-segment basis as well as on a collection of segments. These levels allow legacy systems to be migrated into the COE in stages leading up to Full COE Compliance. In addition, the COE provides a suite of tools for validating COE conformance. For details on compliance, refer to the DII COE I&RTS and the DII COE Style Guide.

2.4.4 Environment Stability

In designing and developing distributed applications, several aspects of the systems environment must be considered to assure a maximum life-cycle for the data (information), the applications (software), and the expertise (human). The issues listed below must be considered when building a durable and robust environment for the design of portable mission applications. At a high level, these issues are:

- C Platform - Hardware/Software -- Operating Environment
- C Standards for OSE Distributed Computing Requirements
- C Vendors/Suppliers' Products -- Availability & Scalability
- C Enterprise Environment -- Constraints
- C Human Factors Engineering and Man-Machine Interface (Graphical User Interface (GUI))
- C Portability and Interoperability
- C Mobility and Mission Considerations

2.4.5 Division of Computing Tasks

To build distributed applications in a client/server environment, it is necessary to determine the division of the computing tasks required and where these tasks will reside (i.e., with the client or the server). The fundamental concept is to analyze the tasks required for the application and logically unbundle the distinct processes that comprise the entire application. For example, a traditional application may have the following components: user interface, application logic, data manipulation or management, data repository (disk device handler), and various data output media. The application logic may be further subdivided into data validation functions and computational logic. These functions may reside individually in separate processors tied together by a communications facility. The communication facility may be a Local Area Network (LAN), a Wide Area Network (WAN), or both. The communications facility is provided by a suite of protocols that can span across network nodes that reside within LANs and WANs.

Figure 2-3 shows the division of tasks between clients (requesters) and servers (responders) or between various servers. This division can be viewed as sequences of requests and responses. The client task can exist independently on its own (self-sufficient), but may require the server to complete a distributed task. The various models of task division may cross other application processes and be shared by cooperating systems. The paradigm of using computers that serve both client and server roles gives way to developing specialized applications that specifically optimize the use of the processor's capabilities and resources. Therefore, the demarcation line of

the division of labor (tasks) may vary between different application components but must remain consistent within a specific application distributed across the network.

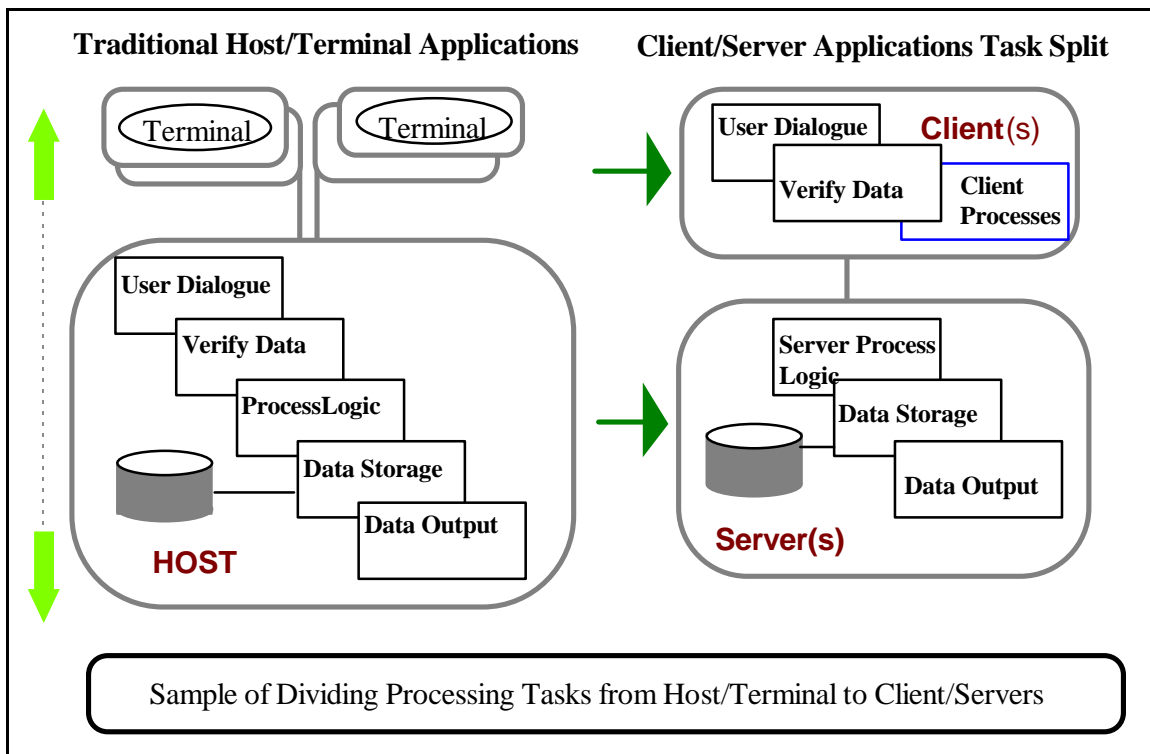


Figure 2-3. Division of Client/Server Tasks

2.4.6 OSE Enabled COE Applications

The TAFIM is based on OSE Guide (Institute for Electrical & Electronic Engineers (IEEE) P1003.0) and is the foundation of the DII COE. OSE application services and protocols enable various mission and business application processes to interface over diverse networks while residing on heterogeneous platforms. The advantage of OSE and DII COE concepts is that the applications themselves provide specific system-level services that may be used selectively with or without operator intervention. The user may be interactive via a user-interface facility (i.e., GUI), another specialized mission, application, or an operational application that needs the OSE services.

For instance, the use of X.400 (Message Handling System) provides the transport mechanism for compound documents or multimedia binary objects using a store-and-forward mechanism. This transport mechanism works like the postal service to relieve the sender of the need to monitor the delivery of the message and its objects to its destination. Similarly, a mission application may choose to send an electronic message (E-mail) automatically in the form of a transaction notification. This is accomplished by inserting a formatted or free-form message directed to another process or entity with assurance of delivery and arrival status of the messages sent.

Mission applications may use some or all of the service facilities of the COE suite. Besides the use of typical E-mail transport, COE application facilities include: network utility-type services for Distributed Directory, File Transfer, Remote Data Access, Transaction Processing, and specialized application layer services such as Remote Procedure Calls (RPC), Open Software Foundation/Distributed Computing Environment (OSF/DCE), Manufacturing Message Services or Information Storage and Retrieval that provide COE-enabled applications across DII.

2.4.7 Distributed COE Applications

The COE application services were intended to be used in OSE as generalized utilities to serve the needs of various mission and enterprise applications. The concept of using OSE applications as generic application utilities has matured and can be used by programmers today by way of functional calls using APIs.

An overwhelming majority of programmer-written applications require only basic communication services: 1) open connection, 2) send and receive messages, 3) close connection, and 4) handle errors. In a connection-oriented mode, the outcome of any communication service request is always acknowledged.

All network services and communications-based application protocols provide utilities and interface services that can be used to support the development of distributed mission applications. The application developer can write applications that take advantage of these services, thus enhancing the customer's application development environment to meet the specific mission requirements. By using these generic network application services, programmer-written applications can provide a strategic advantage to DoD Agencies/Services Information Resource Managers (IRM) in overcoming the backlog of software development.

The distribution of computing tasks as "peer-level" computing services allows software on multiple, independent, heterogeneous processors to cooperate in providing effective delivery of application messages or services. COE provides the foundation services for building the DII and the means to create OSE compliant applications. DII COE provides the ability to work cooperatively across multiple, heterogeneous systems and platforms to accomplish the DoD mission.

2.5 Integration and Runtime Specification

The DII COE I&RTS delineates technical requirements for using the DII COE to build and integrate systems. It provides additional details on many topics covered in this manual and in the DII COE Programmer's Reference Manual. The DII COE I&RTS provides detailed implementation details that describe, from a software development perspective, the following:

- C COE approach to software reuse and data reuse
- C COE runtime execution environment
- C Definition and requirements for achieving COE compliance
- C Process for automated software integration

- C Process for electronically submitting/retrieving software components to/from the COE software repository.

2.5.1 Runtime Environment

The runtime environment encompasses all aspects of the software configuration for the COE. All software and data, excepting low level components of the bootstrap COE, are packaged as segments. Segments are constructed to keep related configuration items together so that functionality may be easily included or excluded. There are six segment types corresponding to the different types of components that may be added to a system. In addition, segments may have attached characteristics, called segment attributes. There are six attributes.

Segment installation is accomplished in a disciplined way through instructions contained in files provided with each segment. These files are called *segment descriptor files* and are contained in a special subdirectory. Installation tools process the segment descriptor files to create a carefully controlled approach to adding or deleting segments to or from the system. The format and contents of the segment descriptor files are the central focus of the runtime environment. Developers are required to adhere to certain procedures to ensure that segments can be installed and removed correctly, and that the installation or removal of segments do not adversely impact another segment.

2.5.1.1 Segment Types

Segment types and attributes are the cornerstone of the COE approach. Developers have considerable freedom in building segments with a few important considerations. The DII COE segment types as defined in the DII COE I&RTS are:

- C COTS Segments
- C Account Group Segments
- C Software Segments
- C Data Segments
- C Database Segments
- C Patch Segments

The segment attributes serve to further define and classify a segment. The DII COE segment attributes as defined in the DII COE I&RTS are:

- C Aggregate Attribute
- C Parent
- C Child
- C COE Component Attribute
- C Web
- C General

2.5.1.2 Segment Prefixes and Reserved Symbols

Each segment is assigned a unique subdirectory called the *segment's assigned directory*. The assigned directory serves to uniquely identify each segment, but it is too cumbersome for use in naming public symbols. Therefore, each segment is also assigned a 1- to 6-character alphanumeric string called the *segment prefix*. The segment prefix is used for naming environment variables and for public APIs and public libraries, where naming conflicts with other segments must be avoided. All segments shall preface their environment variables with the segment's assigned prefix.

The segment prefix is also used to uniquely name executables. All COE component segments shall use the segment prefix to name executables, and it is strongly recommended that all segments follow the same convention. This approach simplifies the task of determining the files that go with each segment and reduces the probability of naming conflicts.

2.5.1.3 Disk Directory Layout

A standardized disk directory structure for all segments is required to prevent two segments from overwriting the same file creating two different files with the same name, or similar issues that cause integration problems. Unfortunately, such problems are often not discovered until the system is operational in the field.

In the COE approach, each segment is assigned its own unique, self-contained subdirectory. A segment is not allowed to directly modify any file or resource it does not "own" (i.e., outside its assigned directory). Files outside a segment's directory are called *community files*. COE tools coordinate modification of all community files at installation time, while APIs attach to the segments that are used at runtime.

The five main types of subdirectories are:

- C Segment Subdirectories
- C Users Subdirectories
- C Developer Subdirectories
- C Test Installation
- C Application - Server

2.5.2 Variants

The exact configuration of segments installed on a workstation is called a *variant*. A variant is simply a collection of segments grouped together for installation convenience. The kernel COE is required to be on each workstation or server, but additional segments are dependent upon the functionality desired and how the workstation or server will be used.

2.5.3 System Management

The COE provides templates for ‘account group segments’ to be used in establishing individual login accounts. Account groups contain template files for defining what COE processes to launch at login time, what functions are to be made available to operators, and preferences such as color selections for window borders. Account groups can also be used to perform a first level division of operators according to how they will use the system. This technique is used in the COE to identify at least five distinct account groups:

- C Privileged Operator (e.g., root),
- C System Administrator,
- C Security Administrator,
- C Database Administrator (DBA), and
- C Non-Privileged Operator.

Additional details on the functions available to each group are in the DII COE I&RTS document however, command-line access for privileged operator accounts is expressly prohibited without prior approval from the DISA Chief Engineer. Within an account group, subsets of the available functionality can be created. These subsets are called *profiles*. An operator may participate in multiple account groups with multiple profiles, and can switch from one profile to another without the need to log out and log in again. Other account groups may exist for specialized system requirements, such as providing a character-based interface, but all account groups follow the same rules.

2.5.4 Development Process

This section describes the COE development process. The concept of automated integration is a powerful feature of the overall development process. This means automated tools are used to combine and load segments, make environmental modifications requested by segments, make newly loaded segments available to authorized users, and identify places where segments conflict with each other. Traditional system level integration then becomes primarily a task of loading and testing segments. Traditional integration tasks are pushed as far down to the developer level as possible.

2.5.4.1 The COE Development Process

The COE approach is designed to be non-intrusive; it places minimal constraints on how developers build, test, and manage software development. It concentrates on the end product and how it will integrate with the overall system. This approach provides the flexibility to allow developers to conform to their internal development process requirements. However, developers are expected to use proven software engineering practices and design tools to ensure robust products. Developers are free to establish a Software Development Environment (SDE) that is best suited to their project. The COE requires only that deliveries are packaged as segments, that segments are validated before submission, and the segments are tested in the COE prior to submission. The DII COE I&RTS provides detailed lists of requirements and suggestions

regarding coding practices, development file structures, and libraries for scripts and files with which developers must be familiar in order to build a successful DII COE Application.

The COE development process consists of the following steps which are described in detail in the DII COE I&RTS.

1. Segment Registration
2. Segment Development
3. Segment Submission
4. Segment Integration
5. Segment Installation

2.5.4.1.1 Segment Registration

Segment Registration is the entry point into the development process. Its purpose is to collect information about the segment for publication in a *segment catalog*. The DII segment catalog is available on-line through a Hypertext Markup Language (HTML) browser and contains information provided by developers in a segment registration form.

There are two steps that constitute the Segment Registration phase:

1. **Register the segment.** The segment registration form can be submitted in written form, through E-mail, or in HTML format.

Once the developer submits the registration form, the information is sent to the process point-of-contact. Segment information is entered into the online repository and the segment catalog with a tentative release date for the segment. The segment prefix and directory requested will be granted unless they have already been assigned to another developer's segment.

2. **Download segments required for development.** When notification is received that segment registration was successful, developers may download COE component segments, developer's toolkit, object code libraries, and other segments required for software development. The DII COE Consolidated Version Description Document and the DII COE I&RTS provide more information on how to download segments, tools, libraries, etc.

2.5.4.1.2 Segment Development

The COE approach is designed to be non-intrusive. Developers are free to establish an SDE that is best suited for their project. The COE requires only that deliveries be packaged as segments, that segments are validated before submission, and that segments are tested in the COE prior to submission.

2.5.4.1.3 Segment Submission

There are two steps in the automated process of submitting a segment. Reference the *Software Documentation Delivery Requirements, Version 1.0* for more information on Segment Specifications.

1. **Compress and encrypt the segment.** The tool *MkSubmitTar* performs this task on a “pre-MakeInstall” formatted segment. The directory *integ* must contain an annotated description of output from *VerifySeg*. When applicable, a test suite must be included for all APIs.
2. **Submit the segment.** The tool *submit* does this electronically across the Internet. Segments submitted via tape must be a relative *tar* of the output from *MkSubmitTar*, not the output of *MakeInstall*. Multiple segments can be delivered on the same tape provided that there is only one segment per physical *tar* tape segment. Tape submission must be used for classified segments or segments which are “very large.”

2.5.4.1.4 Segment Integration

Segments received whether by tape or electronically are tested in isolation and then tested as part of a deliverable system. Validation is performed at each step using the same tool set that the developer used during the development phase.

Prior to submitting a component to DISA, a developer must

- C Package the component as a segment,
- C Demonstrate COE compliance through tools and checklists,
- C Test the segment in isolation with the COE,
- C Provide required segment documentation, and
- C Demonstrate the segment operating within the COE.

From this point on, the process steps are the responsibility of the Software Support Activity (SSA), not the developer. The developer is still an active part of the process in isolating and correcting problems, therefore, the process steps are mentioned.

1. Receive segments.
2. Validate the segment.
3. Submit segment to the online repository.
4. Test segment in isolation.
5. Assign segment DII compliance level.
6. Advance segment to test level.
7. Create configuration definitions.
8. Perform system test.
9. Accept segment.

2.5.4.1.5 Segment Installation

Segments can be distributed to sites either electronically or by other distribution media, as appropriate. The *MakeInstall* tool is used to extract segments and write them to tape or other media. The media is then manually delivered to the site. Installation can also be performed electronically through the *RemoteInstall* tool. The *RemoteInstall* tool operates in either a “push” or “pull” mode. In a push mode, DISA initiates electronic transfer of segments to operational sites. In a pull mode, the remote site initiates the segment transfer. Once received at a site, the site administrator can use the installation tools in the System Administration application to load segments directly onto individual workstations or to multiple workstations as “segment servers.”

2.5.4.2 Migration Considerations

Much of the present and planned functionality of the DII COE is derived from existing legacy systems, not new development, as it is often not feasible to totally abandon a system to start over. Legacy systems must implement a migration strategy that allows them to take advantage of COE benefits. The strategy must simultaneously balance: full COE compliance versus implementation cost; rapid system deployment versus risk to system stability; porting functionality versus new development; and presentation of capabilities users already have versus duplication.

It is helpful to remember that the development approach is to build on top of the DII COE, not to decompose the COE into constituent parts to be integrated with other architectures or software. In other words, the approach is to integrate components from legacy systems into the COE, not to integrate the COE into an existing legacy system. This perspective is fundamental to successful integration.

The key to reusing the COE, and to achieving COE compliance, is the concept of “Public” APIs. APIs represent the linkage through which segments may gain access to COE services. Software developers and integrators must build to public APIs rather than to proprietary versions of software products, since the public APIs will be preserved as the COE evolves. Applications must migrate away from private or legacy APIs since they will not be supported in COE releases.

The following considerations will lead to a successful migration strategy when integrating components from a legacy system into the COE. This information, as well as additional details, may be found in the DII COE I&RTS.

1. Create a requirements matrix.
2. Identify anticipated source code changes.
3. Identify public DII COE APIs to be used.
4. Identify areas where the proposed application overlaps the COE.
5. Identify support services within the legacy system.
6. Develop a schedule for achieving Level 8 compliance (Full COE Compliance Level).
7. Determine how the segment will be integrated with the Executive Manager.
8. Determine which account group(s) the segment will belong to.

9. Determine the required runtime environment extensions.
10. Negotiate new APIs or modifications with the DISA Engineering Office.
11. Build only to public COE approved APIs.

2.5.4.3 Development Environment

The COE development environment allows a variety of the development tools and processes so long as the end products comply with the COE and integrate with the overall system. Even the developers' own internal tools and processes are allowed if the end products do not violate the COE architecture and do not inhibit automated integration, interoperability, and portability.

The technical requirements and specifications of the COE development environment are fully described in the DII COE I&RTS. The following paragraphs provide a brief overview of this information.

2.5.4.3.1 Development Scripts

The development scripts are the scripts that developers use in the COE development environment. Procedures, similar to those suggested in the DII COE I&RTS, must be established to ensure development scripts are kept separate from the runtime scripts.

2.5.4.3.2 Directory Structures

The COE allows developers to create any logical directory structures for use during the development process. However, COE installation tools enforce the COE directory structure as specified in the DII COE I&RTS. Logical directory structures provide benefits not only in software development but also in software maintenance and distribution.

2.5.4.3.3 Private and Public Files

The private files are the files that are accessible only to the COE segment that owns the private files, whereas the public files are open to any COE segments. Developers are expected to hide the process details and data elements of their segments by creating private files and placing high-level functions in the public files. This method minimizes the interdependencies of COE segments, which in turn minimizes the scope of code changes when a segment is changed. This design principle is based on object-oriented programming concepts, e.g., data abstraction and data hiding.

2.5.4.3.4 Developer's Toolkit

The COE provides developers with a toolkit to assist in developing COE compliant applications. The toolkit includes API libraries, C header files, on-line help files, and royalty-and license-free COE development tools. Since the toolkit is not part of the final COE-based system, it is not in the standard COE segment format required by the COE installer.

2.5.5 Database Considerations

Typically, COE-based systems are heavily database oriented. Database considerations are therefore very important in properly architecting and building a system. For more detailed technical information on properly designing databases and database applications, see the DII COE I&RTS. Additional information regarding SHADE can be found in Chapter 5 of this document.

2.5.5.1 Database Segmentation Principles

A COE database server is provided by a COTS RDBMS product. It is used in common by multiple applications, and is a services segment and part of the COE. However, different sites need varying combinations of applications and databases. As a result, databases cannot be included in the Database Management System (DBMS) segment. Instead, these component databases are provided in a database segment established by the developer. The applications themselves are in a software segment, also established by the developer, but separate from the database segment. If the data fill for the database contains classified data, that data fill must be in a separate data segment associated with the database segment.

2.5.5.1.1 Database Segments

The RDBMS is provided as one or more COTS segments. These segments contain the RDBMS executables, the core database configuration, database administration utilities, RDBMS network executables, and development tools provided by the RDBMS vendor. Databases are provided as database segments. These segments contain the executables and scripts to create a database, and tools to load data into the database.

2.5.5.1.2 Database Segmentation Responsibilities

Three groups are involved in the implementation of database segments: DISA, the application developers, and the sites' database administrators. The developers and DISA work together to field databases and associated services for the Database Administrators (DBAs) to maintain. DISA provides the RDBMS as part of the COE. Developers provide the component databases. Sites manage access and maintain the data. Users interact with the databases through mission applications and may, depending on the application, be responsible for the modification and maintenance of data in the databases.

2.5.5.2 RDBMS Tuning and Customization

The core RDBMS instance is configured and tuned by DISA based on the combined requirements of all developers' databases. This allows the RDBMS Server Segments to be reasonably independent of particular hardware configurations and ignorant of specific application sets.

The final tuning of the RDBMS cannot be accomplished until a complete configuration is built and it has an operational load. Developers should provide information for the tuning process, but should not make their applications dependent on particular tuning parameters. Where a

nonstandard parameter is required for operations, developers must provide that information to DISA so the RDBMS services segment can be modified accordingly.

Developers shall not modify the core RDBMS instance's configuration. Extensions or modifications of that configuration require the specific approval of the DISA DII COE Chief Engineer.

If developers modify any of the executable tools (e.g., add User Exits to Oracle*Forms), then the modified version of the tool does not reside with the core database services, but becomes a part of the application's client segment. This prevents conflicts among different modified versions of a core function. The maintenance of that modified tool also becomes the responsibility of the developers.

2.5.5.3 Database Inter-Segment Dependencies

A key objective of the segmentation approach is to limit the interdependencies among segments. Ideally, database segments should not create data objects in any other schema or own data objects that are dependent on other schema. However, one purpose in having a Database Server is to limit data redundancy and provide common shared data sets. This means that there will usually be some dependencies among the databases in the federation. In general, if a schema needs to create data objects in some schema belonging to another segment, those objects will be placed in a separate database segment that modifies the segment that owns those objects. For additional information and an example, see the DII COE I&RTS.

This page intentionally left blank.

3. DII COE Kernel

The COE *kernel* is the mandatory minimal set of software required on every workstation to provide certain common services no matter which workstation will be used. The kernel COE components are the shaded boxes shown in Figure 2-2. The base of the Kernel is a POSIX-compliant vendor-supplied operating system (Windows NT 4.0). This operating system includes a windowing user interface modeled after the X-11 Window standard. Additional commercial products and COE segments are also added to provide required functionality.

3.1 Operating System

The operating system performs the basic functions of resource allocation and processing for external devices attached to the workstation. Additionally, the POSIX compliant operating system provides a set of basic functions that provide the following functionality:

Process Control - Supports the creation, execution, and termination of processes, and provides for the coordination of and communications between processes. POSIX function calls are available to create a new process or to call an executable module; suspend execution until a specified event occurs or for a specified interval; pipe information between processes; extract and manipulate process identification; and terminate a process.

File Operations - Supports the creation, deletion and processing of files and directories. POSIX function calls are available to create files or directories; move through a directory structure; obtain descriptor information regarding a file or check the status of a file; modify the name, status or attributes related to a file; input from or output to a specified position within a file; terminate input/output operations and purge pending operations; remove an existing file or directory from the file system.

Signal Processing - Supports the activation of processes on a timed basis or in response to external events. POSIX function calls are available to manage groups of signals, schedule or wait for signals, examine unprocessed signals, or detect and process blocked signals.

System Parameters - Supports the overall management of configurable system parameters. POSIX function calls are available to determine or set the input and output data rates, or obtain configurable system parameters including the current system date and time.

Group and User IDs - Supports basic security functions inherent in the operating system. POSIX function calls are available to control use of group IDs and associated privileges, or determine and set group or user IDs.

Additional details can be found in the DII COE Programmer's Reference Manual.

3.2 Windowing

In addition to a basic operating system, the COE kernel provides a Windowing environment for the GUI. The standard for this interface is the X-11 Window standard. Windowing implementations include X-Window and MOTIF. Windowing systems provide the following functionality:

Menus - Allows for the generation of custom menus within the desktop environment and the customization of existing applications in the desktop.

Icons - Allows for the creation of pictorial representations of applications associated with the information which the operating system needs to execute the application.

3.3 Desktop

A graphical user environment is provided for all platforms. For UNIX systems, the Common Desktop Environment (CDE) was jointly developed by IBM Corporation, Hewlett-Packard Company, Novell Inc., and SunSoft Inc. It defines a common set of APIs for a CDE that can be implemented in operating environments that can support X-Window desktop and OSF/Motif. The CDE is implemented for UNIX platforms using a commercial product from TriTeal Enterprises. The Desktop helps in bringing every application, network resource, and Internet service into one integrated GUI throughout the systems.

3.3.1 Desktop Tools

The desktop environment provides the following functionality:

Mail Tools - Allows the user to compose, view, and manage electronic mail through the GUI. Allows the inclusion of attachments and communications with other clients through the messaging system.

Calendar Manager - Allows the user to manage, schedule, view appointments, and create a calendar. Meetings and other events can be interactively scheduled with the Mail Tool.

Workspace Manager - Allows the user to organize work into multiple workspaces and navigate between the workspaces.

Editor - Allows the user to create, update, and manipulate text with common functionality including clipboard interaction with other applications.

Terminal Emulator - Allows the user to emulate X-term-like terminals that support terminal emulation of the American National Standards Institute (ANSI) X3.64-1979 and the International Standards Organization (ISO) 6429:1992(E) compliant terminals and to log onto specific host systems.

Calculator - Provides the user with standard calculator functions, using screens and keyboards for inputs and printing.

File Manager - Provides the user with a graphical interface with the file system, including “drag-and-drop” functionality between objects and between cooperating client applications, and a general file type association database.

Print Manager - Provides a simple GUI print job manager to schedule and manage print jobs on any available printer.

Help System - Provides the user with a context-sensitive graphical help system for the desktop.

Style Manager - Allows preferences, such as colors, backdrops, and fonts, to be set interactively for a session using a GUI interface.

ToolTalk - Provides a messaging service that allows communication between applications.

Session Manager - Allows the user to save the current desktop configuration upon logout and restarts the entire session exactly as it was, including applications that were running and the location and size of the windows.

Login Manager - Controls access to the desktop with system password protection and supports password aging and Kerberos.

Application Builder - Provides development tools for constructing applications to run in the desktop environment.

3.3.2 Common Look and Feel

Under the DII concept (refer to the DII COE Style Guide), the standardization of the desktop can increase user productivity, reduce training requirements, and increase the efficiency in the development of individual applications. The commonality in “look and feel” is a key element in the usability of an application and is central to the user interaction with and understanding of a system’s capabilities.

An application can be viewed as the software available to the user to perform a set of related tasks. This software is visible to the user as a collection of window families, each providing the functionality (in terms of objects and information) needed to perform a particular task. In addition, it is possible for applications to share the services provided by a segment when the applications perform common tasks.

Compliance with the specifications contained in the DII COE Style Guide is required in the development of a desktop. All software is expected to comply with the DII COE Style Guide

specifications, with deviations occurring only when called for by operational requirements and approved by DISA.

The DII COE Style Guide provides specifications for the ‘look and feel’ of the software as well as guidance to the design of the system within the general categories of:

- C Input Devices
- C User-Computer Interaction
- C Windows & Window Icons
- C Menus
- C Controls
- C System & Application Design
- C Application Window Design
- C Information Presentation
- C Task Specific Window Design
- C User Support Resources

3.4 System Administration

The System Administration function is used to select and configure printers, manage print jobs, and close windows; reboot or shut down the system, mount file systems, and format hard drives; load or install segments; and change machine ID, edit host information, set system time, configure a workstation as either a client or a server, set routing configuration, and configure mail on a workstation. The System Administrator’s account group sets the System Administrator’s environment to execute the functions listed above.

3.4.1 COE Runtime Tools

The basic set of tools to load and install segments is provided by the DII COE and described in the DII COE I&RTS. These tools provide the System Administrator with the following functionality:

Segment Installer - Allows System Administrator to easily perform the following: execute pre-install script, install segments, execute post-install script, execute de-install script, clone an installation to another workstation with an installed bootstrap COE, install a segment to or from a remote workstation, or update a segment’s home environment to reflect the installed location.

Information Extractor - Allows the System Administrator to view segment information reflecting the installed segment baseline and associated information regarding the installation of new program segments. This tool can be used to display the location and attributes of a named segment, list other segments required to support a specified segment, determine the host name and address of a server for remote installation.

Script Enhancements - Allows for interaction with the System Administrator during the installation process. This tool will prompt the user for textual or Yes/No responses, display informational and error messages, and provide a special prompt for password entry (response not echoed to screen).

Lists - Allows the System Administrator to view associations between segments and authorized users and/or user/groups. The tool lists all accessible segments on a specified server, lists local segments accessible by a specified profile and user accounts assigned to a profile, lists all profiles or users that can access a specified segment, or lists all profiles or segments assigned to a specific user.

3.4.2 Print Services

Each printer is managed by a designated workstation that handles all print requests directed to that printer. Every COE-based UNIX workstation runs a “printer agent” to communicate between the workstation and a printer server. The printer agent provides the following functionality:

Printer Control - Supports an application in controlling printer resources. It will start a print job and establish context (security classification, identification, page characteristics), advance printing operations to the start of the next page, or signal completion of a print job to initiate the actual printing.

Printer Identification - Supports an application in determining the current printing environment. It provides the name, type (ASCII, Postscript, HPCL), and description of the current default printer.

Transfer Data - Provides a method for an application to transfer data to a printer. It sends text data to the default or specified printer, or prints the contents of a file to the default or specified printer.

3.4.2.1 Remote Queue Administration

The information from Process Management is not available at this time and will be provided in an update to this manual.

3.4.2.2 Printer Configuration/Administration

The information for Printer Configuration/Administration is not available at this time and will be provided in an update to this manual.

3.4.2.3 Support for Remote LAN Printing

The information for Remote LAN Printing is not available at this time and will be provided in an update to this manual.

3.4.2.4 Print Service Segments

The information for Print Service Segments is not available at this time and will be provided in an update to this manual.

3.4.3 Process Management

The information for Process Management is not available at this time and will be provided in an update to this manual.

3.4.4 Session Management

The information for Session Management is not available at this time and will be provided in an update to this manual.

3.5 Security Administration

Security of the operating system and COE applications including the development environment is necessary to maintain the integrity of applications and data. The DII COE has established an overall scheme for maintaining overall security and the separation of classified and unclassified data. This scheme is outlined in the DII COE I&RTS. The physical environment in which the system resides is mission-dependent and the responsibility of the individual facility. The following briefly outlines the security measures incorporated into the DII COE and includes on-line system services.

3.5.1 Account Groups

All COE operating environments possess the concept of account groups that are aggregates of users with similar or identical requirements on system resources. Users are normally assigned individual login accounts with configuration files that establish a runtime environment context. These configuration files must be set up and established for each user of the system. An account group segment is a template used within the COE for setting up individual login accounts. Account groups contain template files for defining what COE processes to launch at login time, what functions are to be made available to operators, and preferences such as color selection for window borders.

Account groups can be used to perform a first level division of operators according to how they will use the system. Within an account group, subsets of the available functionality can be created. These subsets are called *roles*. A user may participate in multiple account groups with multiple roles, and can switch from one role to another without the need to log out and log in again. Although other groups may be added as needed, five distinct account groups have been identified within the COE.

Privileged User Accounts - COE operating systems provide a 'super user' account to allow knowledgeable System Administrators to perform special functions and to

troubleshoot. The COE design philosophy severely restricts the use of these types of accounts.

Security Administrator Accounts - Security in the COE is implemented through a Security Server and through a special trusted security client. This client is the Security Administrator application, which is an interface to the Security Server. It allows a Security Administrator to monitor and manage security. Precise functionality of the Security Service is platform-dependent because different approaches have been taken by different vendors to provide security.

The Security Administrator software provides the ability to describe objects (files, data fields, executables, etc.) which are to be protected from general access. This information is used to create profiles that limit an operator's ability to read or modify files. Applications may query the security software to determine the access permissions granted to the current user. The '*Permissions*' file is the mechanism for segments to describe objects and the permissions to grant or deny access to the objects.

System Administrator Accounts - The System Administrator Account Group is a specialized collection of functions that allow an operator to perform routine maintenance operations and load segments.

Database Administrator Accounts - The Database Administrator Account Group is used for performing routine database maintenance activities such as backups, restores, archives, and reloads.

Operator Accounts - An Operator Account is designed for application system users performing mission specific tasks such as creating and disseminating Tasking Orders (TOs), preparing briefing slides, performing *ad hoc* queries of the database, participating in collaborative planning, etc. The precise features available depend upon which mission application segments have been loaded, and the profile assigned to the operator through the Security Manager.

3.5.2 Security Services

The Security Services provide the means to set profile configuration, create or edit local and global user profiles, and create or edit local and global user accounts. The security administrator's account group sets the security administrator's environment to execute the profiles and accounts. The Security Manager provides a facility to update internal profile and user account data structures through command line programs. UNIX-based COEs provide additional functionality over the basic operating system as follows:

Role Management - Controls access to applications through the use of roles in the Operator Account Group. It provides functionality to add or delete users to or from a

specified role, add or delete applications to or from a specified role, change the name of a role, list all available roles, or list applications assigned to a role.

Account Management - Controls the access of individuals to the system. It provides functionality to add or delete a user to or from the system, assign or remove a user to or from a role, add or delete application data to or from a user, or list the roles assigned to a user.

Session Management - Manages parameters related to an individual user's session. It provides functionality to obtain or set the current user's role, display the textual explanation for an error message, or display the user's UNIX identification number.

3.5.3 Security Software

The COE kernel also provides additional software to supplement the basic security provided with the operating system for a given platform:

Screen Locking - Prohibits input from workstations connected and logged on which are not attended.

Secure Window - Opens a secure, read-only console window to display classified output sent to the standard output stream (stdout).

Display Manager - Manages a collection of X-Window displays, prompts for a user ID and password, performs authentication and running of a "session."

Password - Allows users to change their password in the desktop environment.

3.6 Distributed Computing Environment

Distributed Computing Environment (DCE) is an open systems solution that allows users to distribute computing across systems to create a faster, more efficient processing environment. DCE is based on IEEE POSIX standards (Portable Application Standards Committee (PASC) P1003.x). The future of DCE technology is set through open technical forums of the Open Software Foundation (OSF).

The services provided by the COE DCE are shown in Figure 3-1. Core DCE services are provided for all platforms in the DII COE kernel. Extended services vary across platforms and are discussed in Chapter 5.

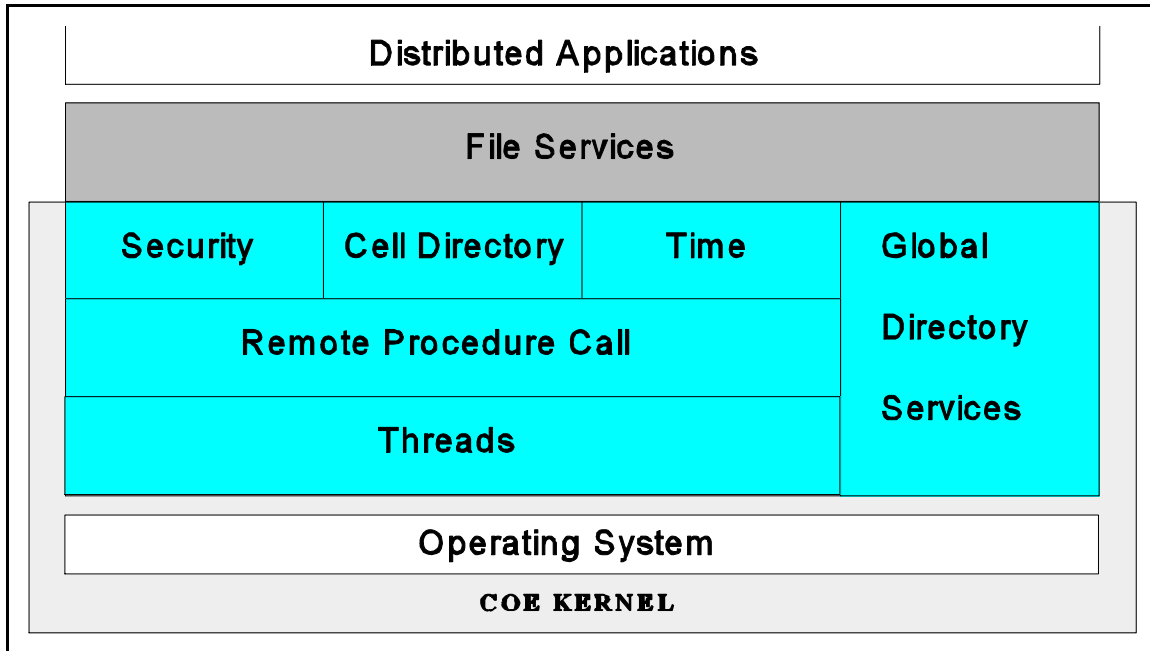


Figure 3-1. Distributed Computing Environment Services

3.6.1 Distributed Computing Environment Cells

A DCE 'cell' is a collection of users, resources, and/or services. Typically within a cell, users make use of computing resources. The resources are provided by the servers. One or more servers may implement a service. A cell may be a collection of machines running DCE services. There may be 3 to 15 Security, Directory, and Time Servers within a cell and approximately 150 clients per server. This is equivalent to having 0 to 25,000 entities per cell. Machines within a cell represent an administrative domain.

3.6.2 Distributed Computing Environment Services

The core DCE provides the following set of services:

Remote Procedure Call - Extends the parallelism provided by threads to allow a procedure to invoke a procedure on another computing resource in a cell. Remote Procedure Call is a facility for the activation of remote processes in a remote system over communications services. Exception handling is built-in while the complexities of network communications are hidden.

Directory Services - Implements a distributed information repository about objects in the computing environment through both the Cell Directory Service and the Global Directory Service. Directory services use both the Domain Name Service (DNS) and X.500

directory services. Attributes are available which describe users, computers, and available distributed services.

Time Services - Provides a service essential for application synchronization and time value standardization in a controlled transaction environment. It provides scalability through a hierarchical propagation scheme and a fault-tolerant fall-back configuration, and can import time from a variety of sources such as the National Time Pendulum (NTP), World Wide Web (WWW), Spectracom, and Global Positioning System (GPS).

Security Services - Provides for a 'Single,' network-wide registry with the ability to support single sign-on capability for security, database access, and verification of credentials. Service is scalability through the use of replication, client caching, and inter-cell operations. It provides robust services by way of authentication through Privileged Attribute Certificates and POSIX Access Control Lists. This is done both using intra-cell and inter-cell security mechanisms. The service also provides for message encryption services using Data Integrity and Privacy. It supports powerful delegation support capabilities.

Threads - Allows independent execution threads to be created within the same program, allowing for parallel execution of multiple computing tasks with minimal overhead.

3.6.3 Distributed Computing Environment Supplemental Services

The DII COE supplements the COTS DCE product with additional tools to assist the developer to create segments that use DCE and to install and manage DCE at the operational site. Tools specifically designed for the DII COE rules for DCE applications include:

- C Standard Server Installation
- C Standard Server Initialization
- C Standard Client Binding
- C Standard Reference Monitor
- C DCE Verification
- C Template Application

Additional details are available in the DII COE I&RTS.

4. DII COE Developer Toolkit and API's

This section gives an overview of the COE Tools and Application Program Interfaces (API). It includes the main functionalities of these tools and their basic uses. More detailed information may be found in the DII COE Programmer's Reference Manual.

4.1 COE Tools

One of the strongest features of COE is its wide collection of tools. These tools can be subdivided into two categories: 1) the runtime tools which are delivered in the kernel with the software code, and 2) the development tools which are software tools for developers to use. The runtime tools are described in Chapter 3, DII COE Kernel. The developer's toolkit is used to perform the following functions:

- C Build installation segments,
- C Test the segments, and
- C Manage segments.

4.2 Application Program Interface

APIs provide software applications access to the platform services or to the COE component services. A programmer invokes specific services in their application software by using API calls. Standards-based API calls as specified in the DII COE provide a level of application portability and independence from the underlying platform services.

4.2.1 Concept

A common definition of an API is:

An API specifies the mapping between program syntax and the features of a specific service, and thereby provides access to that service from applications written in a particular programming language, when the application is bound to the service by the language implementation. (IEEE - PASC POSIX)

An alternate short definition of API is: "An interface is a boundary between two entities for purposes of processing data." For example, an API is the interface between mission application software and the underlying application platform services. The platform is defined as a set of organized resources and services on which the application software will run, and provides specific services required by the application software.

An aspect of the use of APIs is related to the way a service is called. The API call and attributes specified must be bound to a programming language and linked to the program in an executable mode in order to work. A specific language, such as COBOL or C, can have native verbs that call a macro function of the system or network service desired. This is the easiest way to develop programs for the business user. However, a verb or procedure call function from a supplier's

product line may lock the program in that environment. Therefore, the API calls (verbs) within programming languages must also be standardized and tested to allow the program to be portable across platforms.

Currently high-level APIs, protocol-specific verbs, or calls for programming languages that will perform the functions across platforms are not available. However, work is ongoing to make the APIs more user-friendly, allowing application programmers to use macro services as utilities in developing new kinds of programs.

It is absolutely vital that developed applications access services only through common COE-approved APIs. The use of private or proprietary (i.e., undocumented or not COE-approved) APIs make component upgrades difficult and cause mission applications, that use those private APIs, to fail or to malfunction.

4.2.2 Role of APIs in Application Development

The access to systems and network services by DII application segments or other mission applications requires the use of program service calls using APIs. In application development, developers use APIs to request the services of the common system components such as directories, file transfers, E-mail, Remote Database Access, etc. The service calls provide access to the specific Application Service Entities (ASE) that are accessible by the appropriate application program linkages to the programming languages' binding services. The APIs are the means of accessing the service elements of all DII network and platform services.

The Developer's Toolkit of the DII COE includes APIs to access services to:

- C Help in the development of new applications.
- C Help developers provide information to users and solicit input. These are intended to assist when new applications (segments) which have a broader applicability for developers are installed or removed from a system.
- C Print data.
- C Provide mapping functions.
- C Use the Distributed Computing Environment (DCE) for building distributed applications using a client/server model.

The standards-based APIs are documented in the DII COE Programmer's Reference Manual and the various DII COE documents that are part of the COE Toolkit.

The APIs for the Common Desktop Environment (CDE), Oracle, and Sybase are not included in the toolkit but are provided, supported, and documented by their respective vendors/developers.

The Developer's Toolkit is provided separately, **not** as a DII COE-compliant segment. Details on how to load the tools onto a DII COE-based system are provided in the toolkit documentation. Refer to the Installation Instructions for the *Developer's Toolkit* provided in the DII COE Condolitated Installation Guide.

4.3 Standards

The DoD has adopted existing standards and developed special standards that apply to the COE. These standards and the goal of portability affect the development of applications designed to run in the COE.

The framework for major service areas of the TAFIM applies to the DII COE. Conversely, the DII COE is compliant with the TAFIM's classification of standard services.

The CASE tools and environments specific to COE tools are compliant with the principles of the Portable Common Toolkit Environment (PCTE). PCTE is the International Standards Organization/International Electrotechnical Commission (ISO/IEC) 13719 Standard used with C and Ada Language bindings, commonly referred to as European Computer Manufacturers' Association (ECMA) 149 (PCTE).

Additional standards applicable to this document are related to the SDE described in the IEEE 1209 (Evaluation and Selection of CASE Tools), DoD-STD-1467 (Software Support Environment), National Institute of Standards and Technology - European Computer Manufacturers' Association (NIST-ECMA) 500-211 (Reference Model for SEE Frameworks), and MIL-HDBK-782 (Software Support Environment Acquisition).

With respect to software life cycle processes, the MIL-STD-498 *Software Development and Documentation*, and the ISO/IEC 12207 *Software Life Cycle Process* standards are applicable. For the GUI behavioral design, the MIL-STD-1801 *User Computer Interface* applies.

Standards for APIs are primarily based on Portable Operating Services Interfaces (IEEE POSIX P1003.n series). Additionally API standards created by various industry and consortia are adopted as "Public" APIs. These are X/Open (XPG4 Guide), Open Software Foundation (OSF/DCE), Network Management Forum (NMF), and other expert groups such as X.400 API Association (XAPIA) that feed the API standards process into the formal standards developing organizations.

"Look and Feel" standards, contained in the DII COE Style Guide, are required for the development of a desktop application. All software is expected to comply with the DII COE Style Guide specifications, with deviations occurring only when called for by operational requirements and approved by DISA.

This page intentionally left blank.

5. DII COE Infrastructure Services

DII COE Infrastructure Services are a set of capabilities provided to support mission applications and systems operations on an as-required basis. Infrastructure Services expand and supplement the basic services provided in the Kernel and are installed at the discretion of a site administrator. The DII COE Programmer's Reference Manual contains information about each of the applications discussed in this section. Infrastructure Services of both commercial and government products are usually not available (or tested) on all DII COE platforms, especially Windows® NT. The DII COE Programmer's Reference Manual and applicable Version Description Documents should be checked for availability.

5.1 Management Services

DII COE management services extend the basic Kernel capability for system administrators to manage the ongoing operation of their system. These services tend to be commercial packages designed to extend services on a standard UNIX platform. They have been adapted and tested for use within the COE.

5.1.1 System Management Services

System management services provide a comprehensive set of services for administering systems, expanding system administrators' effectiveness by permitting common system administration functions to be performed remotely. It simplifies management of a client/server network of heterogeneous workstations and other desktop systems. The COE provides Tivoli's Management Environment which is compliant with the Object Management Group's Common Object Request Broker Architecture (CORBA). The COE provides Tivoli for both clients and servers that consist of the base Tivoli and extensions.

5.1.1.1 Management Environment

The Tivoli Management Environment is a basic set of system administrator tools.

Administrator management - Provides a mechanism for explicitly identifying system administrators and the system administration resources the administrator is allowed to access, as well as the privileges allowed when accessing those resources. System administrators can perform administration functions without requiring the root password.

Notification groups - Provides a log of administrator activity separated into application-specific notification groups. These notices provide an administration-specific audit trail.

Profiles and profile managers - Manages common system resources (e.g., user accounts, host namespace, etc.) according to a resource template, called a profile. A profile includes the definition of rules that will be enforced for the profile attributes, such as the user password criteria.

Basic managed node - Permits Tivoli to perform remote management actions on client nodes, including querying its physical and logical configuration without having to log into that node.

Policy regions - Establishes default policy, invoked when a new instance of the resource is created (e.g., user). Performs policy validation, used to check that existing resources still comply with the rules.

Interconnection - Interconnects managed resources to provide scalable system administration and to reduce the effects of individual management server failure.

Tasks and jobs - Define administrative procedures for one or more platforms, permitting common system administration tasks, such as removing core files, pruning out unused user accounts, or checking for invalid user passwords, and perform security audits.

Scheduled actions - Perform certain actions at regular intervals or at off-hours so that they do not interfere with mission-critical operations.

5.1.1.2 Administration

Extends the basic Tivoli profile capability to provide user, group, and host namespace management. Tivoli Admin also provides basic and extended Network Information Service (NIS) domain management. Tivoli Admin's main functions are as follows:

User account management - Supports adding, editing, and deleting user accounts, as well as distributing user account information to other profile managers, to managed nodes, and to NIS and Extended NIS (NIS+) domains.

Group management - Supports adding, editing, and deleting UNIX groups, as well as distributing group information to other profile managers, to managed nodes, and to NIS and NIS+ domains.

Host namespace - Supports adding, editing, and deleting host name-Internet Protocol (IP) assignments, as well as distributing host namespace information to other profile managers, to managed nodes, and to NIS domains.

NIS - supports adding, editing, and deleting NIS maps and map data, as well as making and pushing NIS maps.

NIS+ - Supports the distribution of user and group profiles in an NIS+ environment.

5.1.1.3 Software Distribution

Tivoli Courier provides software distribution (and removal) across heterogeneous platforms with a wide number of options, including repeater capability to reduce traffic over wide-area networks (local fan-out), single source to many destinations in a single action, and pre- or post-processing.

5.1.1.4 Monitoring

Tivoli Sentry provides a secure, distributed, general purpose system-monitoring capability. It does not require polling to detect changing conditions on clients, as Tivoli Sentry monitors run locally, that is, on the managed node being monitored. Sentry also implements role and privilege definitions to control which administrators are permitted to define or modify Sentry monitors. When specified conditions are detected, Sentry can be configured to take corrective action automatically, send E-mail, notify specific administrators, or run an executable.

5.1.2 Network Management Services

Network management services provide capabilities for an authorized user to manage server and network activity from a client computer.

5.1.2.1 Management Information Base (MIB) Functionality

The Internet community has defined sets of standards for networking which compose various Management Information Bases (MIBs). These standards start as Requests for Comments (RFCs) and consist of definitions of measurable quantities or items of information. These quantities are broken down into groups. Some of these standards concern quantities useful in a Network Management function. Network Manager Systems implement these MIBs by collecting and monitoring these information sets in addition to setting specifically identified elements.

5.1.2.1.1 Host Resources MIB

The host resources MIB consists of 6 groups of information designed for managing host systems on a network:

System Group - Information relating to system startup, system time and running time, and numbers of users and processes.

Storage Group - Parameters relating to local storage areas (e.g., disk partitions and file systems) and current usage information.

Device Group - Identifies and provides status on all installed devices, processors, ports, disk drives, network interfaces, etc.

Running Software - Identifies applications currently running on the system and provides details including type of process, calling sequence and parameters, path, and status.

Running Software Performance - Provides processor time and memory being used to run the application.

Installed Software - Identifies (name, description, version, type, and date installed) software packages installed by the COE segment installer. Also, identifies the current version of the operating system and installed patches.

5.1.2.1.2 Network Management MIB

The Network Management resources MIB consists of 10 groups of information that constitute a simple architecture and system for monitoring Protocol/Internet Protocol (TCP/IP) based networks:

System Group - Information on network entities including an identification of the network management subsystem, a description, network name, location, contact, and provided services. The time since the last restart is also provided.

Interfaces Group - Detailed information about each network interface in a managed network entity including a description, status, performance, current throughput, and error rate.

Address Translation Group - Functionality transferred to other groups. Retained for compatibility with earlier standards.

IP Group - Detailed information regarding entity processing of IP datagrams, a table containing the entity's IP addressing information, and a table containing information for each network route known to the entity, and a table to map IP addresses to physical addresses.

ICMP Group - Detailed information regarding entity processing of Internet Control Message Protocol (ICMP) messages.

TCP Group - Operating parameters of the entity and dynamic information regarding any open TCP connections.

UDP Group - Detailed information regarding User Datagram Protocol (UDP) datagrams and a table of UDP Listeners (processing application) accepting UDP datagrams.

EGP Group - Information regarding Exterior Gateway Protocol (EGP) messages and a table of the entity's EGP neighbors.

Transmission Group - Detailed statistics dependent on the physical method used to transmit information.

SNMP Group - Detailed information on the processing done by a Simple Network Management Protocol (SNMP) entity.

5.1.2.1.3 Remote Network Management MIB

The Remote Network Management MIB extends the Network Management MIB to allow for the remote management of a network. The MIB allows a probe to collect statistics continuously even when communication with the remote manager is not possible or efficient. The probe will attempt to notify the manager when exceptional conditions exist. The MIB consists of 9 groups of information that define objects for managing remote network monitoring devices:

Statistics Group - Statistics measured by probes for each monitored interface.

History Group - Periodic statistical samples from the network stored for later use and to provide an operational baseline.

Alarm Group - Statistical samples of probe variables that are compared to thresholds to signal significant deviations from normal operations.

Host Group - Statistics associated with host systems identified on the network.

HostTopN Group - Prepares reports describing hosts that top a list ordered by one of the available host parameters.

Matrix Group - Statistics for conversions between sets of two addresses.

Filter Group - Processes captured packets that match a filter equation. Captured packets can generate events.

Packet Capture Group - Permits the capture of packets after they flow through a channel.

Event Group - Generates events and provides notification of events from the monitoring device.

5.1.2.2 COE Functionality

The COE currently provides 3 commercial packages for network monitoring which supplement the basic functions provided by the Kernel:

Empire System Manager Agent - Implements the Network Management and the Host Management MIBs. Additionally, a UNIX-specific extension to the Host Management MIB is implemented.

NetMetrix Remote Monitoring Agent - Implements the Remote Network Management MIB and a toolbox containing a graphic toolset.

Streams - Provides a general facility and a set of tools for the development of UNIX communications services.

5.1.3 Print Management Services

Print management services provide enhanced functionality for network printers. The COE includes a SUN product, Newsprint, which provides network printing support and utilities and print file filters and fonts.

5.2 Communications Services

COE communications infrastructure services provide the capability to interact with other DoD message and data systems.

5.2.1 COE Communications Software / USA Comm Server

The COE Communications Software (CS) provides reliable message delivery through Tactical Communications Interface Modules (TCIMs), Tactical Radio, LANs, and WANs. The Communications Server supports both DCE and non-DCE network environments. The CS enables clients without TCIMs to send/receive messages through TCIMs attached to other servers. The COE CS provides the common communications infrastructure for the Army Common Hardware and Software program.

5.2.2 Unified Build (UB) Comms Service

The Comms service takes coded input from various tactical and message traffic inputs and decodes the input for use by other UB services. Input may be from Link 11, Link 14, navigational (GPS type) interfaces, and other special interfaces. These services are discussed in Section 6.4, DII COE Support Applications, Correlation.

5.3 Data Management Services

DII COE data management services provide an application-independent capability to maintain and administer data. In the DII COE, these services are provided by a COTS database management system (DBMS).

5.3.1 SHADE Concepts

Many current systems are not truly interoperable due to data redundancy, inconsistency, and structures that are not shareable. The SHADE approach is meant to provide an architectural structure that solves the data sharing issues. This in turn should guarantee data consistency, eliminate redundancy, and promote data interoperability.

SHADE uses database segmentation and Shared Data Servers (SDSs) as the primary underlying mechanisms to enable data sharing.

The DII COE I&RTS and the DII COE I&RTS Database Development and Integration Standards documents discuss database concepts and the DBMS design environment for the DII COE client-server architecture.

5.3.2 Database Management Systems

The COE currently supports 4 DBMS's. Although each commercial product has some unique features, use of these features is discouraged since it does not support application portability. The COE is working towards the concepts of SHADE. Currently the COE includes the following commercial products:

- C Oracle
- C Sybase
- C Informix
- C Access

5.3.3 Developer Constraints

Database developers must conform to the COE database server environment, thereby limiting the likelihood of data corruption. The Shared Data Server (SDS) configuration and developer's implementations must ensure that each user's connection to a database:

- C Functions in it's proper context
- C Does not interfere with any other user's connection

Developers will no longer have exclusive use of the DBMS available to them. This consideration must be taken into account since the DBMS is a control service that supports all applications' databases and is not an application-specific data management tool.

5.3.4 Database Integration

Developers must implement their database so that dependencies on a particular vendor's DBMS product is limited, since the SHADE Database Server provides shared data management. The following requirements will help ensure a smooth integration:

Evolutionary Implementation:

Database developers can support incremental development by maintaining modularity within their component databases. A component database must coexist without corrupting other data. Care must be taken to not create unintended dependencies on a specific DBMS. Vendor-specific features should be avoided.

Database Segmentation:

A database should be subdivided into coherent segments. A database supporting only one application may use a single unique segment, while a database supporting multiple applications may need to utilize shared and unique segments. Utilizing shared segments can support the data requirement of mission applications without carrying extra services.

Multiple Databases:

The COE database architecture is one where the component databases share DBMS resources. Individual database segments may be changed without reference to others only if they are unique segments. Developers are responsible for maintaining their own data access and update rules.

Data Integrity:

A COE DBMS must provide features that support data integrity to protect the information stored. To preserve database integrity, the database server must:

- C Prevent connections between an application and data that belongs to any other application.
- C Ensure the recoverability of failed transactions of a crashed system.
- C Enforce the developer's integrity constraints when they are defined within the database.

Discretionary Access:

Access to databases must be discretionary because not all users have the same permissions to use applications. Each user-application connection will have only the permissions needed for that context. DII databases are characterized as public (usually read-only), or private (discretionary access). The three components of discretionary access include:

- C Session Management
- C Discretionary Access Control
- C Access Management

Client/Server Independence:

Developers must preserve the independence of their applications so that applications are not built such that they are required to reside on the SDS.

Distributed Databases:

COE DBMS products must support data being spread across multiple sites. Data may be replicated or fragmented to improve responsiveness and increase availability. The objective is to provide location transparency, meaning that the user does not need to know where the data resides.

Backup and Recovery:

DBMS and database backup and recovery is the responsibility of the site's system and database administrators. COE databases must be developed with the constraints of the DII COE tools and DBMS vendor tools.

5.3.5 Creating Database Objects

The objective when creating database segments is to maintain consistency across different databases and improve independence. The characteristics of the following objects should be taken into account when a database is under development:

Database Accounts:

There are three types of accounts within COE with different functions and levels of access:

- C Database Administrator (DBA)
- C Database Owner (DBO)
- C Users

Physical Storage:

Developers cannot assume that DII SDS will have uniform configurations or that they will remain static.

Definition Scripts:

The developer is responsible for creating a shell script that contains all database definition commands for a specific object.

Database Objects:

The definition of a database schema (objects, constraints, rules, etc.) is the responsibility of the developer. Objects include:

- C Database Tables
- C Data Elements
- C Database Views
- C Rules
- C Indexes

Database Roles:

A role is a group of access privileges for objects which should be created by the developer.

Grants:

Grants provide permission on objects that allow users to access data they do not own. Developers should only grant the minimum set of permissions needed for access.

5.3.6 Inter-Database Dependencies

An object is considered a dependent object if it references any other objects as part of its definition. Upon creation of a dependent object, all of its references must be resolved.

5.4 Network Services

The COE supplements the Kernel network services available to the user. Currently the COE implements DCE extended services that build on the DCE core provided in the Kernel and a suite of Internet/WWW applications.

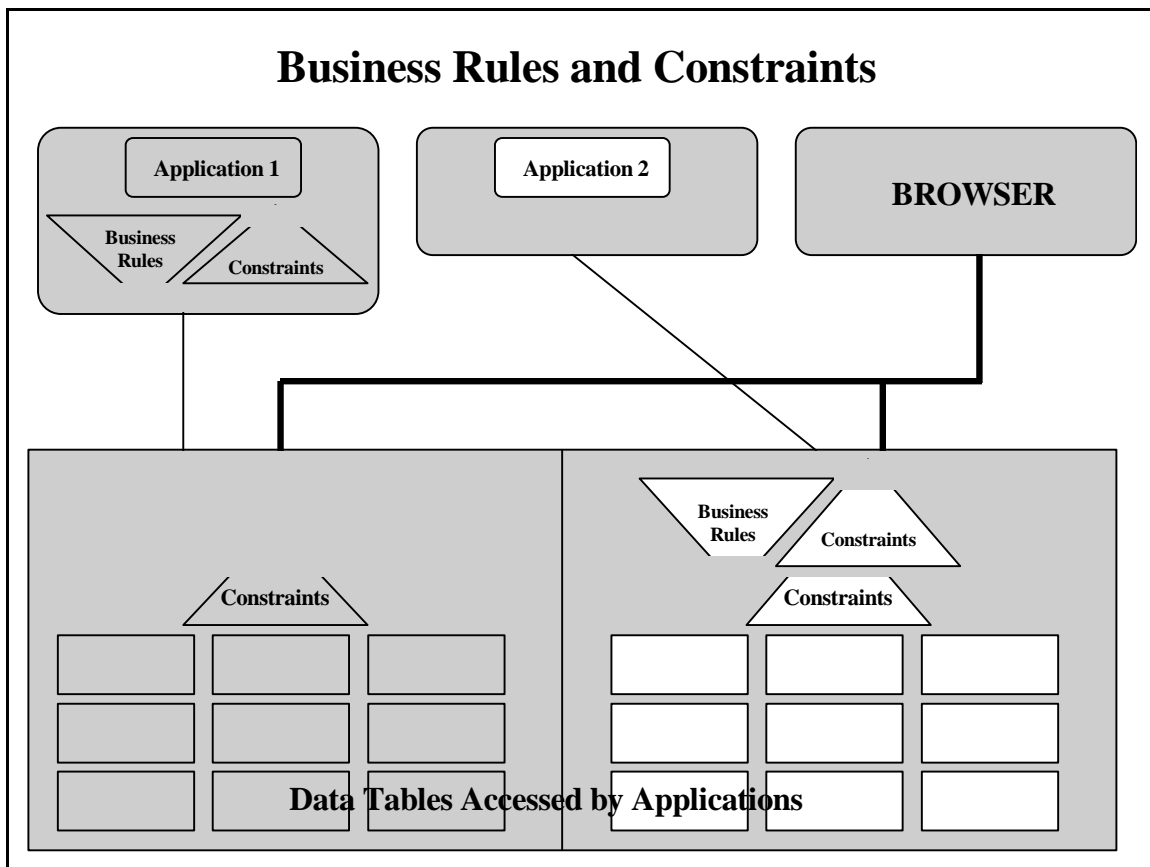


Figure 5-1. Business Rules and Constraints

5.4.1 Distributed Computing Environment

The concept of the Distributed Computing Environment was introduced in Section 3 since the client portion of DCE resides on every workstation. However, to configure a DCE cell and activate a distributed environment, one or more servers must be configured. This application is contained in the infrastructure utilities. At least one server is required to support the DCE. The

server directly supports the Cell Directory Service (CDS), the Security Service, and the Distributed Time Service (DTS).

5.4.1.1 Cell Management

The COE also provides a utility which enables graphical management of a DCE cell. The DCE Cell manager by Chisholm Technologies provides the following set of tools:

Namespace Manager - Views, searches, and edits the CDS.

Security Manager - Views, searches, and maintains the security registry.

Configuration Manager - Monitors and manages DTS and host status.

5.4.1.2 Distributed File Service

The Distributed File Service (DFS) is an enhancement to the DCE built on top of the core DCE services provided in the Kernel. DFS extends the core to support file sharing and data management activities. DFS enables collections of workstations to share files as a single unit.

5.4.2 Internet Services

Access to the Internet has become an essential mission requirement for many organizations. The COE provides a set of utilities to launch the user into the Internet and the World Wide Web:

Web Browser - HTML-capable World Wide Web browser from Netscape.

Web Server - Utilities for establishing and maintaining home pages and a World Wide Web server from Netscape.

News Server - Utilities for establishing and maintaining discussion groups on a server from Netscape. Discussion groups can be private (designated members) or public.

News Make Group - Information is not available at this time but will be provided in a later release of this manual.

5.5 Utilities

Utilities provide an additional set of application-independent services in support of DII COE mission applications, infrastructure services, and authorized users.

5.5.1 General Utilities

General utilities provide generic capabilities to COE platforms.

5.5.1.1 File Related Utilities

The COE provides the following basic utilities to provide enhanced functionality for users in obtaining and storing data:

File Transfer Protocol (FTP) - General purpose utility that allows transfer of files to or from remote computers. FTP can be set up as anonymous or require a specific user logon sequence.

File Compression - General purpose file compression scheme (GZIP) which reduces the space used by a data file without losing any information contained in the file. Similar to the ZIP function used on PCs.

5.5.1.2 UNIX Script Utilities

UNIX script utilities are interpretive languages that provide additional functionality over standard UNIX command line utilities:

Practical Extraction and Report Language (PERL) - PERL is optimized for scanning text files, extracting information, and generating reports. Primarily used for systems management by systems administrators.

Tool Command Language (TCL) - TCL is a script language with aspects of procedural languages with access to an associated graphical tool kit. Used for simple programs which require a graphical user interface.

5.5.1.3 Windows Emulation

Windows Applications Binary Interface (WABI) is a Sun solution to integrating MS Windows® applications into the Solaris environment. It requires the purchase of MS Windows® 3.11 and the actual application. WABI does not guarantee that all applications will operate correctly. Information about how to check WABI/application compatibility is in the DII COE Programmer's Reference Manual.

5.5.2 Security Utilities

COE security utilities fall into two broad classes: probes that assist the security administrators in preventing unauthorized access and sentries that attempt to prevent or detect unauthorized access.

5.5.2.1 Probes

Probes are utilities used to detect security weaknesses and vulnerabilities that can be used to penetrate a system. COE provides the following utilities:

Crack - A compute-intensive, configurable utility that attempts to guess passwords. Guessable passwords are exploitable holes in system security. Freeware versions of Crack are available.

Security Administrator Tool for Analyzing Networks (SATAN) - Systematically probes an Internet host for known security weaknesses. SATAN is distributed as Freeware.

Security Profile Inspector (SPI) - A collection of security tools and utilities. Contains tools to attempt to expose weak file or group structures, test operating system vulnerabilities, or crack passwords. Detection tools use check sums to detect system authenticity and patch currency, or changes in files, users, and groups. A utility is provided which allows inquiries into security objects. SPI is available only to the government.

5.5.2.2 Sentries

Sentries are usually passive routines that detect the actions of a probe or a system penetration. The COE provides the following utilities:

Courtney - Released by the government coincident with the release of SATAN to detect possible SATAN probes on a host. Courtney is available in the public domain.

TCP Wrappers - Monitors external requests for common network services such as FTP or TELNET. Can provide additional security by limiting these services to specific workstations. Shareware versions of TCP wrappers are available.

Tripwire - Detects possible intrusions and assesses damage by comparing electronic file signatures. Tripwire is also public domain.

This page intentionally left blank.

6. DII COE Support Applications

DII COE support applications are an upper level layer of applications software and toolkits designed to provide specific functionality to an end user or to support the development of an application or process. Support applications are tailored to user functions which will be performed at a workstation. Support applications consist of commercial and Department of Defense (DoD) products. Availability of applications may vary according to hardware platform. Additional details on availability and functionality are contained in the DII COE Programmer's Reference Manual.

6.1 Office Automation

Office automation is a collection of general purpose desktop capabilities that enhance end user productivity. The current office automation application:

Internet Relay Chat - Provides users the ability to electronically interact with other users in realtime. The service allows sharing of messages between group members as they are typed.

6.2 Mapping, Charting, Geodesy, & Imagery

The MCG&I service is provided in the Joint Mapping Toolkit (JMTK) software package. The service provides the ability to graphically present information to the operator. The JMTK provides the following functionality:

Manipulate Symbols - Provides the ability to alter the characteristics of an object or a family of objects. Function calls include the ability to:

- C Modify an object family's display attributes, data field, fill characteristics, or visibility
- C Copy display attributes into a template
- C Animate an object
- C Retrieve attribute or coverage information about an object
- C Set a parent object's display attributes, data field, fill characteristics, or visibility
- C Change selected physical characteristics of an object.

Draw Objects - Provides the ability to create and alter graphical objects. Function calls include the ability to:

- C Create text, polygon, or polyline objects through animation
- C Draw an arc, box, character (including 16 bit symbols), circle, ellipse, line, polygon, polyline, rectangle, sector, line segment, slash, symbol, or text string
- C Include a bitmap in a drawing

- C Alter the orientation of a line segment
- C Change the text in textual objects.

Control Display Settings - Provides the capability to alter attributes specific to text objects. Function calls include the ability to:

- C Modify an object family's font, line style, line type, or line width
- C Apply a template to an object or object family
- C Set an object's font, line style, line type, or line width
- C Set the highlight color
- C Change the color of an object or object family to the highlight color.

Control Display Features - Provides the capability to alter features and symbols on a map. Function calls include the ability to:

- C Add a feature or list of features to a map
- C Add a point to a polygon or polyline
- C Change the color of an object or object family
- C Sets whether or not an object or object family is selectable
- C Create an object through animation
- C Create an object attributes template
- C Draw a weather line segment (front)
- C Set the overall intensity, foreground color, and background color of a geographic display.

Edit Features - Provides the capability to move an object from a window, remove an object, or modify features on a map. Function calls include the ability to:

- C Remove classes of features or products from the library
- C Modify attributes of a feature or list of features on a map
- C Move an object on a map
- C Remove a feature from a map
- C Remove a list of map products from a display
- C Remove an object from an object list.

Transform Data - Provides the capability to perform conversion to standard units. Function calls include the ability to:

- C Extract projection data from a given window
- C Determine the latitude and longitude of a point on a geographic display
- C Determine the location on a display given latitude and longitude.

Conversion is provided between radians, degrees, nautical miles, miles, kilometers, and feet.

Manage Windows - Provides a consistent Windows interface to the Chart Manager from client applications. Function calls include the ability to:

- C Initialize and terminate the interface
- C Request ownership of a feature product
- C Request rendering responsibility for a map product
- C Abort a draw command
- C Add an object to a list or class
- C Add a product or list of products to a map
- C Change displayed maps and features
- C Create an object class or list
- C Create or terminate a map window
- C Erase an object family or list of objects
- C Draw a world view map in a window
- C Transfer an object to another class or list
- C Obtain the X Window ID of a geographic display
- C Map a window to the screen to allow display
- C Terminate JMTK operations on a window
- C Reorder map products in a display
- C Hide a window
- C Get and register a channel to a window.

Adjust Display View - Provides capabilities to alter the view of an existing map. Function calls include the ability to:

- C Magnify the current view (zoom in)
- C Center the display on a specified map point
- C Change the scale of a map
- C Change the mode of the cursor and the left mouse button
- C Set the boundary attributes or width of a geographic display.

Query Spatial Database - Provides functionality to use the Spatial Database (SDB). Function calls include the ability to:

- C List available Spatial Databases (SDBs)
- C Connect to a SDB
- C List available maps
- C Define or retrieve the Area Of Interest (AOI)
- C Locate search path information related to a specific data type within an AOI
- C Manage search path information
- C Process SDB matrix data
- C Return error messages associated with an error code
- C End the connection to the SDB.

Manage Display - Provides general display management services. Function calls include the ability to:

- C Abort object animation
- C Manage input data sources
- C Manage interval timer services
- C Transfer buffered requests to the chart manager
- C Flush event queues or extract an event from a queue
- C Count pending map events
- C Change the order within a queue
- C Manage point select focus
- C Send events to other processes using a window
- C Set control keys for animation.

Transform Coordinates - Provides conversion between geodetic, universal transverse Mercator, and military grid reference map coordinate systems. Function calls include the ability to:

- C Translate coordinates between the 3 systems
- C Translate coordinates to and from text.

Manage Symbol Library - Provides the capability to use reference map features. Function calls include the ability to:

- C Operate map reference lists
- C Describe the list of features drawn into a map window.

Terrain Analysis - Provides capabilities to make tactical decisions based on the geographic display. Function calls include the ability to:

- C Determine area of coverage for weapons fanning
- C Determine line of site characteristics.

Process Errors - Provides error processing capabilities for the Chart Manager and draw module components of the JMTK. Function calls include the ability to:

- C Process chart manager errors for clients
- C Handle errors that occur in the draw module
- C Indicate to the chart manager that an error has occurred in a specific window.

Provide Information - Allows end users to extract information from a geographic display. Function calls include the ability to:

- C Provide the distance and bearing between 2 points
- C List available chart manager features

- C List display features or attributes for a map window
- C Extract information about a map window.

Manipulate Symbols - Provides Chart Manager parsing routines. A function call returns the next matching option on a command line.

Communicate with Windows - Manages the communications channel to the Chart Manager. Function calls include the ability to:

- C Identify features or products to the draw module
- C Attach a draw module to a chart manager
- C Initialize a draw module, communicate between the chart manager and a draw module
- C Manage a communications channel
- C Interface with Xwindow
- C Provide socket information about a communications channel
- C Manage events
- C Terminate chart manager execution.

Manage Memory - Provides capabilities to manage memory and search functions. Function calls include the ability to:

- C Obtain and release memory
- C Provide the current map search path.

6.3 Messaging

Message Handling capabilities in the COE are provided by the Common Message Processor (CMP). It provides the functionality required to prepare, receive, analyze, and validate United States Message Text Format (USMTF) and Army Variable Message Format (VMF) messages. The CMP is also able to normalize message data into formats usable by other applications.

6.3.1 User Services

The CMP provides a user interface to access message handling functions and built-in journaling functions. Messages can be interactively created and edited in a distributed environment. The CMP's analysis capabilities support the filtering of message traffic and assigning them to separate windows based on parameters extracted from the message such as recipient, time stamp (date-time group), precedence, and classification. Messages may be generated automatically from operational databases.

6.3.2 Application Support

The core of the CMP is in two standalone software systems which handle the two basic aspects of message handling. Additional modules perform support functions. Rules for message processing

are contained in a set of tables that must be configured. CMP provides the following functionality:

Message Processing - Analyzes and controls the processing of inbound message traffic received from Communications Services (see Section 5.3). Function calls are available to obtain an inbound message, determine type of processing required, journal a message, profile a message to determine interest level, validate a message to ensure that sets and fields comply with rules established for subsequent processing, parse a message to extract information, and pass extracted information to another location or process (i.e., additional processing, database posting, viewing, etc.).

Message Generation - Creates a user- or computer-generated message and sends it to the communications services area for transmission. Function calls are available to manually generate a message according to a template, automatically generate a message according to a template using information passed from an automated process, determine detailed addressing data for the message header, provide for electronic coordination prior to message release, and route the completed message to a user with message release authority for dissemination.

Message Handling Support - Performs general support and common services for message processing. Function calls are available to configure CMP components, process errors external to the CMP which impact message handling, audit classification aspects of message processing, normalize specified formats to USMTF, translate to and from VMF, combine multi-section input messages and section output messages, and annotate messages being coordinated.

Message Journaling - Provides an interface to the systems journal of all incoming and outgoing messages. Function calls are available to modify a journaled message and retransmit, search all received messages, and list or display all or selected released messages.

6.4 Alerts

The Alerts service builds on the signal processing functions in the Kernel (see Section 3.1) to manage tactical processing on a workstation or, with an alerts server, on a family of workstations. The Alerts service defines alert and event message formats and controls the posting, reviewing, queuing, and dequeuing of alerts and events. Events can be generated by a process or generated by a system event. The Alerts service provides:

Alert Generation - Provides a function call that instructs the alerts server to generate a specific alert.

Alert Definition - Provides an interactive interface to create and name routing definitions that include addressing by role, duty group, or user ID.

6.5 Correlation

The Correlation service is part of the Unified Build (UB) software package. The service maintains a consistent tactical picture across a theater of operations based on inputs provided by service components. The service associates high-level tactical objects (ships, submarines, and aircraft), fixed and mobile installations, and land force units with tracking data. Tracking data can be obtained from Electronic Intelligence (ELINT) sources, Tactical Digital Information Links (TADIL), and Joint Surveillance Target Attack Radar System (JSTARS)-type Ground Moving Target Indicator (GMTI) radar tracks.

6.6 Situation Display

The Situation Display service is part of the UB software package. The service displays the current tactical picture across a theater of operations based on the information stored in the Track Database. The track data, representing correlated and uncorrelated tactical elements, is presented on graphic situation displays and made available for tactical decision aids that support decision making.

This page intentionally left blank.

7. DII COE User Information

This section provides basic user information and covers the DII COE standards in terms of applicable reference documents available to the COE users. A complete bibliography of DII COE documentation is available in Appendix C.

The DII COE reflects the principles, architecture, and standards adopted by TAFIM which includes the Portable Common Toolkit Environment (PCTE) and the ongoing work in the Integrated Software Engineering Environment (ISEE) that are also related to the Portable Operating Systems Interface (POSIX).

7.1 POSIX Calls

POSIX originated in the Institute of Electrical and Electronics Engineering (IEEE) Standard Committee for the Portable Operating Standards Committee (PASC) and refers to a Portable System Interface for compliant computer environments.

There are three parts to the implementation of the standard:

- C Definition of terms, global names and concepts
- C Interface facilities
- C Data interchange format.

A list of POSIX function calls (APIs) can be found in the DII COE Programmer's Reference Manual. They are part of the interface and are required for use in the programming environment to effect portability to assure that programs can interface reliably with any POSIX-compliant operating system using the calls specifications.

7.2 Reference Documentation

The following standards are applicable directly to the DII COE:

- C POSIX.1 is the Portable Operating System Interface that is currently a Federal Information Processing Standard (FIPS) 151-2 based on ISO/IEC 9945-1 international standards and born from IEEE Portable Operating System Committee (POSC) P1003.1 (ANSI) Standard.
- C POSIX.2 is part 2 of the Shell and Utilities is currently the FIPS 189 based on ISO/IEC 9945-2 international standard and born from IEEE POSC P1003.2 (ANSI) Standard.

7.3 Extending the COE

Some segments may require extensions to the COE or extensions may be desired to obtain additional functionality. In this case, refer to the DII COE I&RTS. It describes techniques for

making desired common extensions to the COE, such as adding menu items or icons to the desktop. It also describes techniques for making common types of extensions and techniques for addressing less frequently encountered extensions that may be necessary in the user's environment.

7.4 Problem Reporting

DISA has established a procedure to report any problems detected in the DII COE and has established an Incident Control Center to process incoming problems. Problems are either recorded on a Global System Problem Report (GSPR, DISA form 291) or transmitted via electronic mail.

Once received, the DISA Configuration Management (CM) department assigns a control number to the problem and provides the number to the originator prior to document review. The problem is then sent to a system engineer for evaluation. If the problem has been fixed, CM is notified and the update is distributed. Otherwise, the problem is assigned for action. When completed, CM is notified and the updates propagated as before.

The procedure for end-user notification of the disposition of the problem reported is under consideration and will be published when established and approved for dissemination.

7.5 COE On-line Services

The DII COE also features a look ahead at the future of the technology including new development tools by providing a set of on-line services. These Internet- and intranet-based services allow rapid communication and development of strategic plans, requirements analysis, and documentation. They also help in the timely development of, dissemination of, and enterprise-wide access to information and segments. These services offer a wide variety of information-representation, viewing, and processing functions across the major corporations, governmental agencies, and global enterprises making up the COE community. For more details on the COE on-line services, see the DII COE I&RTS. Additional information can be found by consulting the Internet Website addresses (Universal Resource Locator(URL)) below:

<u>Document or Area</u>	<u>Location</u>
Auxiliary Services	Netscape homepage (http://www.netscape.com)
DII COE Documentation	DISA website (http://www.disa.mil)
Distributed Computing Environment	Transarc homepage (http://www.transarc.com:80/afs)
JMTK Documentation	DISA website (http://www.disa.mil)

7.5.1 Security Features

The COE on-line services can be divided into classified and unclassified systems. The classified system is accessible only via the Secure Internet Protocol Router Network (SIPRNet), while the unclassified system is openly accessible on the Internet. Both types of systems use secure operating systems, databases, and network software and maintain auditing of all system access and of all sensitive operations. Classified and unclassified components reside on physically distinct computer systems. The classified system is further protected by three layers of security consisting of firewalls, user authentication, and encryption.

7.5.2 COE Information Server (CINFO)

The COE information server (CINFO) provides information to the COE community via the WWW and SIPRNet. Non-sensitive general information is provided by the unclassified WWW site. The classified site on SIPRNet contains more detailed information, including a list of all available segments, the segment versions, and all segment patch information, including the information on upcoming system changes and special installation instructions.

This page intentionally left blank.

Appendix A: Acronym List

ANSI	American National Standards Institute
AOI	Area Of Interest
API	Application Program Interface
APP	Application Portability Profile
ASE	Application Service Entities
C4I	Command, Control, Communications, Computers, and Intelligence
CCB	Configuration Control Board
CDE	Common Desktop Environment
CDS	Cell Directory Service
CINC	Commanders in Chief
CINFO	COE Information Server
CM	Configuration Management
CMM	Capability Maturity Model
CMP	Common Message Processor
COE	Common Operating Environment
CORBA	Common Object Request Broker Architecture
COTS	Commercial Off-the-Shelf
CS	COE Communications Software
CSCI	Computer Software Configuration Item
DBA	Database Administrator
DBMS	Database Management System
DBO	Database Owner
DCE	Distributed Computing Environment
DFS	Distributed File Service
DII	Defense Information Infrastructure
DISA	Defense Information Systems Agency
DNS	Domain Name Service
DoD	Department of Defense
DTS	Distributed Time Service
ECMA	European Computer Manufacturers' Association
EGP	Exterior Gateway Protocol
ELINT	Electronic Intelligence
FIPS	Federal Information Processing Standard
FTP	File Transfer Protocol
GMTI	Ground Moving Target Indicator
GOTS	Government Off-the-Shelf
GPS	Global Positioning System
GSPR	Global System Problem Report

GUI	Graphical User Interface
GZIP	General purpose file compression scheme
HTML	Hypertext Markup Language
ICMP	Internet Control Message Protocol
IDB	Integrated Database
IDD	Interface Design Document
IEC	International Electrotechnical Commission
IEEE	Institute for Electrical & Electronic Engineers
IP	Internet Protocol
IRCC	Internet Relay Chat Client
IRM	Information Resource Managers
ISEE	Integrated Software Engineering Environment
ISO	International Standards Organization
JIEO	Joint Interoperability and Engineering Organization
JMTK	Joint Mapping Toolkit
JOPEs	Joint Operations Planning and Execution System
JSTARS	Joint Surveillance Target Attack Radar System
JTA	Joint Technical Architecture
LAN	Local Area Network
MIB	Management Information Base
MCG&I	Mapping, Charting, Geodesy, and Imagery
NIS	Network Information Service
NIST	National Institute of Standards and Technology
NMF	Network Management Forum
NTP	National Time Pendulum
OM	Operator's Manual
OSD	Office of the Secretary of Defense
OSE	Open System Environment
OSF	Open Software Foundation
PASC	Portable Application Standards Committee
PCTE	Portable Common Toolkit Environment (ECMA)
PERL	Practical Extraction and Report Language
POC	Point of Contact
POSC	Portable Operating System Committee (now PASC)
POSIX	Portable Operating System Interface for UNIX
PSA	Principal Staff Assistant

RDBMS	Relational Database Management System
RFC	Request for Comments
RPC	Remote Procedure Calls
SATAN	Security Administrator Tool for Analyzing Networks
SDB	Spatial Database
SDS	Shared Data Server
SDE	Software Development Environment
SEI	Software Engineering Institute
SHADE	Shared Data Environment
SIPRNet	Secure Internet Protocol Router Network
SNMP	Simple Network Management Protocol
SPI	Security Profile Inspector
SSA	Software Support Activity
TADIL	Tactical Digital Information Links
TAFIM	Department of Defense Technical Architecture Framework for Information Management
TCIM	Tactical Communications Interface Module
TCL	Tool Command Language
TCP/IP	Transmission Control Protocol/Internet Protocol
TO	Tasking Order
UB	Unified Build
UDP	User Datagram Protocol
URL	Universal Resource Locator
USMTF	United States Message Text Format
VMF	Army Variable Message Format
WABI	Windows Applications Binary Interface
WAN	Wide Area Network
WWW	World Wide Web
XAPIA	X.400 API Association

This page intentionally left blank.

Appendix B: Glossary

Account Group: A template for establishing a runtime environment context for individual operators. Account groups are typically used to do a high-level segregation of operators into system administrators, security administrators, database administrators, or mission-specific operators.

Affected Account Group(s): The account group(s) to which a segment applies. Functionality provided by the installed segment will normally appear to the operator as new menu items or icons in the affected account group(s).

Aggregate Segment: A collection of segments grouped together, installed, deleted, and managed as a single unit.

Application Program Interface (API): The API is the interface, or set of functions, between the application software and the application platform. An API is categorized according to the types of service accessible via that API. There are four types of API services: 1) User Interface Services, 2) Information Interchange Services, 3) Communication Services, and 4) Internal System.

Approved Software: Commercial software products that have been tested as compatible with the COE. In this context, approved software implies only that the software has been tested and confirmed to work within the environment. It does not imply that the software has been approved or authorized by any government agency for any specific system.

Bootstrap COE: That subset of the COE that is loaded in order to have enough of an operational environment so that segments can be loaded. The bootstrap COE is typically loaded along with the operating system through vendor-supplied instructions or UNIX commands such as *tar* and *cpio*.

Client: A computer program, such as a mission application, that requires a service. Clients are consumers of data while servers are producers of data.

Commercial Off-The-Shelf Software (COTS): Software that is available commercially. Examples include versions of UNIX, X Windows, or Motif, as well as approved software such as Oracle, Sybase, and Informix.

Common Operating Environment (COE): The architecture, software infrastructure, core software, APIs, runtime environment definition, standards and guidelines, and methodology required to build a mission application. The COE allows segments created by separate developers to function together as an integrated system.

Community Files: Files that reside outside a segment's assigned directory. To prevent conflict among segments, community files may be modified only through the COE installation tools. Examples of community files include: */etc/passwd*, */etc/hosts*, */etc/services*.

Compliance: A numeric value, called the compliance level, which measures the degree to which a segment conforms to the principles and requirements defined by COE standards, and the degree to which the segment makes use of COE services. Compliance is measured in four areas, called compliance categories. The four categories are Runtime Environment, Architectural Comparability, Style Guide, and Software Quality.

Component Database: Individual databases within a multi-database design.

Configuration Control Board (CCB): The organization responsible for authorizing enhancements, corrections, and revisions to the COE or to a COE-based system.

Database: A structured set of data, managed by a DBMS, together with the rules and constraints for accessing the data.

Database Management System (DBMS): Software to manage concurrent access to shared databases.

Database Schema: The design of a particular database.

Descriptor Directory: The subdirectory *SegDescrip* associated with each segment. This subdirectory contains descriptors that provide information required to install the segment.

Descriptors: Data files (contained in the segment's descriptor directory) that are used to describe a segment to the COE. The software installation and integration process uses descriptor directories and their descriptor files to ensure COE compliance. Descriptor files permit automated integration and installation.

Development Environment: The software environment required to create, compile, and test software. This includes compilers, editors, linkers, debug software, and developer configuration preferences such as command aliases. The development environment is distinct from the runtime environment, and must be separated from the runtime environment, but is usually an extension of the runtime environment.

Distributed Database: A database whose data objects exist across multiple computer systems or sites.

Distributed Processing: The ability to perform collaborative processing across multiple computers. This capability allows processing load to be distributed.

Environment: In the context of the COE, all software that is running from the time the computer is rebooted to the time the system is ready to respond to operator queries after operator

login. This software includes the operating system, security software, installation software, windowing environment, COE services, etc. The environment is subdivided into a runtime environment and a Software Development Environment (SDE).

Environment Extension File: A file that contains environmental extensions for the COE. Segments use extension files to add their own environment variables and other items to the COE.

Fragmentation Schema: The distribution design for a distributed database.

Government Off-The-Shelf (GOTS) Software: Software developed through funding by the U.S. Government.

Kernel COE: That subset of the COE component segments which is required on all workstations. As a minimum, this consists of the operating system, windowing software, security, segment installation software, and Executive Manager.

Multi-Database: A collection of autonomous databases.

Profile: The subset of the total COE-based functionality that is to be made available to a group of individual operators.

Runtime Environment: The runtime context determined by the applicable account group, the COE, and the executing segments.

Segment: A collection of one or more CSCIs (Computer Software Configuration Items) most conveniently managed as a unit. Segments are generally defined to keep related CSCIs together so that functionality may be easily included or excluded in a variant.

Segment Prefix: A 1-6 alphanumeric character string assigned to each segment for use in naming public segments.

Server: A computer program that provides some service. Servers are producers of data while clients are consumers of data.

Service: A function that is common to a number of programs, such as performing an extensive calculation or retrieving a category of data.

Session: An individual connection between an application program and a database management system.

Shared Data Server (SDS): A DII compliant platform that provides data server functions for multiple mission applications.

Superset: The sum total collection of all COE-based segments available to the development community. The superset includes the COE as well as all mission-application segments.

Variant: A subset of the superset of all software. This subset includes the COE and is fielded to service an operational mission area. A variant represents that collection of segments, including COE component segments, that are suitable for a particular site, mission area, or workstation. See also the definition of mission area, site, system, and workstation variants.

Workstation Variant: A collection of segments as installed and configured on a particular workstation.

Appendix C: Reference Documents

Table C-1. List of Referenced Documents

NOTE: The following documents are NOT included with this release set of documentation but are available from DISA CM.

Reference Document Title	Version/Date
<i>Design, Requirements, and Specifications</i>	
Technical Architecture Framework for Information Management (TAFIM)	Version 4.0
Architectural Design Document for the Defense Information Infrastructure (DII) Common Operating Environment (COE)	Draft January 1996
Defense Information Infrastructure (DII) Common Operating Environment (COE) Version 2.0 (Series) Baseline Specifications	June 28, 1996
User Interface Specifications for the Defense Information Infrastructure (DII)	Version 2.0 Preliminary Draft December 31, 1995
Defense Information Infrastructure (DII) Common Operating Environment (COE) Integration and Runtime Specification	Version 2.0 October 23, 1995
	Version, 3.0 Draft January 1997
Configuration Management Software and Documentation Delivery Requirements	Version 1.0 14 June 1996
Interface Design Document (IDD) for the Common Operating Environment (COE) Communications Software	Rev. A.1 LL-412-04-01 November 27, 1995
Software Requirements Specification for the Common Operating Environment (COE) Communications Software	Rev. A.1 LL-412-10-01 November 27, 1995

Table C-2. COE Documentation Cross-Reference

NOTE: The following documents are included with this release set of documentation.

Guides and Manuals		
DII COE Version 3.0.0.6 Programming Guide (Windows NT 4.0)	CM-400-04-14 February 5, 1997	See Appendix E
APIs		
DII COE Version 3.0.0.6 Application Programmer Interface (API) Reference Guide (Windows NT 4.0)	CM-400-80-03 February 5, 1997	See Appendix D of the Programmer's Reference Manual
DII COE Consolidated Version Description Documents		
DII COE Version 3.0.0.6 Version Description Document for the Kernel (Windows NT 4.0)	CM-400-49-10 February 5, 1997	
DII COE Version 3.0.0.6 Version Description Document for the Developer's Toolkit (Windows NT 4.0)	CM-400-51-08 February 5, 1997	
Version Description Document MS-Office Version 1.0.0.0/4.2c (MS-Windows NT 3.51)	LL-413-02-01 (no date)	
DII COE Consolidated Installation Guide Documents		
DII COE Version 3.0.0.6 Kernel Installation Guide (Windows NT 4.0)	CM-400-47-07 February 5, 1997	
Internet Relay Chat Client Application Installation Procedures - Quick Guide	LL-400-66-01 August 21, 1996	
Microsoft Office 4.2c Application Installation Procedures - Quick Guide	LL-413-03-01 October 3, 1996	
Consolidated System Administrator's Guide Documents		
DII COE Version 3.0.0.6 System Administrator's Guide (Windows NT 4.0)	CM-400-46-06 February 5, 1997	

Appendix D: List of JTA Mandated Standards

This appendix lists the technology areas where the Joint Technical Architecture (JTA) mandates specific standards that may be applicable where they correspond to the DII COE Components. The mandated standards are described in detail in throughout the JTA sections and further summarized in a set of tables in an appendix for quick reference. Please, consult the JTA documentation for additional technical specification or see the electronic links to the appropriate DISA organization to obtain JTA documentation by accessing their Web Site. Note that the standards references listed below may apply to future segments of the DII COE.

JTA Mandated Standards applicable to DII COE. See JTA documentation for detailed list.

- C CDE User Interface Services
- C MOTIF
- C Data Management Services
- C Document Interchange
- C Web Technology
- C Graphics Data Interchange
- C Images on the Web
- C Geospatial Data Interchange
- C Imagery Data Interchange
- C Sound Data Interchange
- C Video Data Interchange
- C Atmospheric Data Interchange
- C Oceanographic Data Interchange
- C Graphic Services
- C Operating System Services
- C Internationalization Services
- C Remote Procedure Computing
- C Distributed Object Computing
- C Host Standards
- C Electronic Mail
- C Directory Services
- C Domain Name System (DNS)
- C File Transfer
- C Remote Terminal
- C Network Management
- C Network Time
- C Bootstrap Protocol (BOOTP)
- C Dynamic Host Configuration Protocol (DHCP)
- C Hypertext Transfer Protocol (HTTP)
- C Uniform Resource Locator (URL)
- C Connectionless Data Transfer
- C Transmission Control Protocol (TCP)
- C User Datagram Protocol (UDP)

C Internet Protocol (IP)OSI/Internet Interworking Protocol
C Video Teleconferencing (VTC) Standards
C Analog Facsimile Standards
C Digital Facsimile Standard
C Dissemination Standards
C Router Standards
C Internet Protocol (IP)
C Interior Routers
C Exterior Routers
C Local Area Network (LAN)
C Point to Point Standards
C Combat Net Radio (CNR) Networking
C Integrated Services Data Network (ISDN)
C Asynchronous Transfer Mode (ATM)
C 25kHz Service
C 5kHz DAMA Service
C 25kHz TDMA/DAMA Service
C Data Control Waveform
C Earth Terminals
C Phase Shift Keying (PSK) Modems
C Low Data Rate (LDR)
C Medium Data Rate (MDR)
C Automated Link Establishment
C AntiJamming Capability
C Data Modems
C Very High Frequency (VHF)
C Ultra High Frequency (UHF)
C Super High Frequency (SHF)
C JTIDS/MIDS Transmission Media
C SONET Transmissions
C Activity model
C Data Model
C DoD Data Definitions
C J-Series Family of Message Standards
C US Message Text Format (USMTF) Messages
C Database to Database Exchange
C Activity model
C Data Model
C DoD Data Definitions
C J-Series Family of Message Standards
C US Message Text Format (USMTF) Messages
C Database to Database Exchange
C General TAFIM
C OSF Style Guide
C Commercial Style Guides

- C DoD HCI Style GuideDomainlevel Style Guides
- C Application Software Entity Security Standards
- C Operating System Services Security
- C Security Auditing and Alarms Standards
- C Authentication Security Standards
- C Host Security Standards
- C Security Algorithms
- C Security Protocols
- C Evaluation Criteria Security Standards
- C Internetworking Security Standards
- C HumanComputer Interface (HCI) Security Standards

This page intentionally left blank.

Appendix E: Programming Guide for Windows NT 4.0

Table of Contents

Preface	81
E.1 Writing Programs Using the COE Tools	81
E.1.1 Overview	81
E.1.2 Additional Sources of Information	82
E.2 Application Development Overview	82
E.2.1 Writing Your Application with the DII COE APIs	82
E.2.2 Building Your Application with the DII COE APIs	83
E.2.2 Running Your Application	83
E.3 Segment Development	83
E.3.1 Segment Layouts	84
E.3.2 COE Tools Overview	85
E.3.2.1 Running the COE Tools From the Command Line	85
E.3.2.2 COE Runtime Tools	85
E.3.2.3 COE Developer Tools	85
E.3.3 Building Your Segment	86
E.3.3.1 Identifying and Creating Required Subdirectories	86
E.3.3.2 Creating and Modifying Required Segment Descriptor Files	86
E.3.3.3 Installing a Segment	89
E.3.4 Customizing Your Segment	89
E.3.4.1 Adding Menu Items	90
E.3.4.2 Adding Icons	96
E.3.4.3 Reserving a Socket	97
E.3.4.4 Displaying a Message	97
Attachment A - Sample Segments	99
AA.1 Sample Account Group Segment Layout	99
AA.2 Sample Software Segment Layout	101
Attachment B - Verifying Segment Syntax and Loading a Segment onto a Floppy Diskette ..	103
AB.1 Running VerifySeg Against the Sample Segment	104
AB.2 Running TestInstall Against the Sample Segment	104
AB.3 Running TestRemove Against the Sample Segment	105
AB.4 Running MakeInstall Against the Sample Segment	105
Attachment C - Installing the Developer's Toolkit	107

List of Tables

Table E-1. Segment Descriptor Files	87
Table E-2. SegInfo Descriptor Sections	88

List of Figures

Figure E-1. Segment Directory Structure	84
---	----

Preface

The following conventions have been used in this document:

[HELVETICA FONT]	Used to indicate keys to be pressed. For example, press [RETURN].
Courier Font	Used to indicate entries to be typed at the keyboard, Windows NT commands, titles of windows and dialog boxes, file and directory names, and screen text. For example, execute the following command: A:\setup.exe
"Quotation Marks"	Used to indicate prompts and messages that appear on the screen.
<i>Italics</i>	Used for emphasis.

E.1 Writing Programs Using the COE Tools

E.1.1 Overview

This document provides an introduction to the capabilities of the Defense Information Infrastructure (DII) Common Operating Environment (COE) tools for the DII COE Version 3.0.0.6 for Windows NT Operating System Version 4.0. These tools consist of a set of runtime tools and a set of developer's tools.

This document has been designed to help developers start using the DII COE tools. This document explains the basic use of the tools, regardless of whether they are run from a menu or from the command line.

The document consists of the following sections and appendices:

Section/Attachment	Page
Application Development Overview Provides an overview of how to write, build, and run an application.	82
Segment Development Discusses the different types of segments and the process of segment creation.	83
Sample Segments Describes how to install sample segments, which can be used to test segment installation and execution.	99

Section/Attachment	Page
Verifying Segment Syntax and Loading a Segment onto a Floppy Diskette Provides examples of how to convert a segment to the <i>DII COE Integration and Runtime Specification</i> segment format, verify segment syntax, temporarily install and remove a segment, and load a segment onto an installation floppy diskette.	103
Installing the Developer's Toolkit Describes how to load the Developer's Toolkit, which contains the components needed to create segments that use DII COE components.	107

Descriptions assume familiarity with the C programming language and with the Windows NT development environment.

E.1.2 Additional Sources of Information

Reference the following documents for more information about the DII COE toolkit:

- C *Defense Information Infrastructure (DII) Common Operating Environment (COE) Integration and Runtime Specification* Version 2.0, DII COE I&RTS:Rev 2.0, Inter-National Research Institute, October 23, 1995
- C *Defense Information Infrastructure (DII) Common Operating Environment (COE) Version 3.0.0.6 Application Programmer Interface (API) Reference Guide (Windows NT 4.0)*, DII.3006.NT40.RG-1, Inter-National Research Institute, February 5, 1997.

E.2 Application Development Overview

Developers may require access to public Application Programmer Interfaces (APIs) in order to ensure an application complies with the *DII COE Integration and Runtime Specification*. To use these public APIs, developers must (1) include the public `include` files with the DII COE tools header and (2) compile and link the application with the Developer's Toolkit `include` directory and libraries. Public APIs are documented in the *DII COE API Reference Guide (Windows NT 4.0)*.

E.2.1 Writing Your Application with the DII COE APIs

To access the DII COE tools through the provided APIs, you must include the following header in your application:

```
#include <DIITools.h>
```

The standard location for the Developer's Toolkit header is:

```
DII_DEV\include
```

E.2.2 Building Your Application with the DII COE APIs

To build your application with the DII COE APIs, you must link your application with the `COECom.lib`, `COESeg.lib`, `COETools.lib`, and `COEUserPrompts.lib` libraries, which are on the DII COE Developer's Toolkit floppy diskettes.

The standard location for the Developer's Toolkit libraries is:

```
DII_DEV\libs
```

To compile the application, make sure the `include` file path in the compile environment includes `<local path>\DII_DEV\include`. Also make sure the library (`lib`) path includes `<local path>\DII_DEV\libs`.

E.2.2 Running Your Application

The DII COE provides the foundation and infrastructure in which one or more applications run. Applications must be formatted properly as segments to operate under the COE. The segment is the basic building block of the COE runtime environment. A segment is a collection of one or more Computer Software Configuration Items (CSCIs) that are managed most conveniently as a unit. Segments generally are defined to keep related CSCIs together so functionality easily may be included or excluded. All applications must be put in the DII COE runtime environment segment format to be installed onto a DII COE-compliant machine.

Once an application has been put in the proper segment format, the segment can be installed in a disciplined way through instructions contained in files provided with each segment. These files are called segment descriptor files and are contained in a special subdirectory, `SegDescrip`, which is called the segment descriptor subdirectory. Installation tools process the segment descriptor files to create a carefully controlled approach to adding or deleting segments to or from the system.

Once installed, your application can be invoked in the DII COE environment in two ways: (1) running your application from a command shell window or (2) invoking your application from an icon. The easiest way to test your application is to invoke it in a command shell window. This gives you easy access to your application for debugging purposes and allows you to check any diagnostic information your application is generating. Section 3.4, *Customizing Your Segment*, describes how to set up your application to be invoked as a menu item or as an icon.

E.3 Segment Development

The following subsection discusses the different types of segments and the process of segment creation. Reference Section 5.0, *Runtime Environment*, of the *DII COE Integration and Runtime Specification* for a more detailed explanation of segments.

E.3.1 Segment Layouts

In the DII COE approach, each segment is assigned a unique, self-contained subdirectory. DII COE compliance mandates specific subdirectories and files underneath a segment directory. These subdirectories and files are shown in Figure 1. Six segment types exist: Account Group, COTS (Commercial Off-the-Shelf), Data, Database, Software, and Patch. The precise subdirectories and files required depend on the segment type. Some of the subdirectories shown in Figure 1 are required only for segment submission and are not delivered to an operational site.

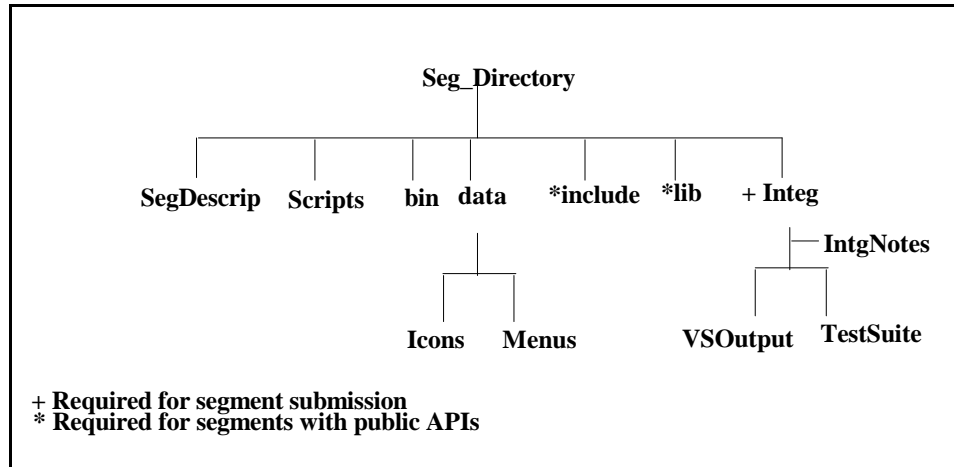


Figure E-1. Segment Directory Structure

The following runtime subdirectories are normally required, depending on the segment type: (1) *SegDescrip*, which is the directory containing segment descriptor files; (2) *bin*, which is the directory containing executable programs for the segment; and (3) *data*, which is the subdirectory containing static data items, such as menu items, that are unique to the segment, but that will be the same for all users on all workstations.

The *SegDescrip* directory is required for every segment because it contains the installation instructions for the segment. A segment cannot modify files or resources outside its assigned directory. Files outside a segment's directory are called community files. COE tools coordinate modification of all community files at installation time. Reference Section 5.5, *Segment Descriptors*, of the *DII COE Integration and Runtime Specification* for a detailed explanation of *SegDescrip* files.

E.3.2 COE Tools Overview

The COE tools were constructed to aid developers in the creation and ultimate installation of DII COE segments. All tools can be run from the command line.

E.3.2.1 Running the COE Tools From the Command Line

This section provides a brief overview of running the COE tools from the command line. Reference the following sources for detailed information about the COE tools: Appendix C, *COE Tools*, of the *DII COE Integration and Runtime Specification*; the Developer's Toolkit release notes; the on-line help files for the tools; and the help page provided by the tools.

When run from the command line, the tools are designed to run interactively and accept one or more command line parameters.

The tools are used to communicate with the outside world in two ways. First, the tools use the exit function to set the DOS status variable. The status return value is set to 0 for normal tool completion or to 255 if an error occurs. A status return value greater than 0 but less than 255 indicates a completion code that is tool specific. The `ERRORLEVEL` function may be used to examine a tool's exit status.

Second, the tools use `stdin` and `stdout` and thus support input and output redirection. Redirecting `stdin` allows the tools to receive input from a file or from another program, while redirecting `stdout` allows the tools to provide output to other programs.

NOTE: Redirecting `stdin` is not always convenient. The `-R` command line parameter allows a tool to read input from a response file instead of from `stdin`.

For example, the following statement can be used to write the results of `VerifySeg` to a file:

```
VerifySeg -p d:\testsegs Seg1 > results.txt
```

E.3.2.2 COE Runtime Tools

Reference Appendix C, *COE Tools*, of the *DII COE Integration and Runtime Specification* for a complete description of the DII COE runtime tools.

E.3.2.3 COE Developer Tools

Windows NT tools are delivered on a floppy diskette and may be copied to any directory desired for development. The `TestInstall` and `TestRemove` tools must be run as `Administrator` because they modify files the user may not own. `VerifySeg` should also be run as `Administrator`, although it is not mandatory. The `VerifySeg` tool requires the user to have write permission to the segment against which the tool was executed.

Reference Appendix C, *COE Tools*, of the *DII COE Integration and Runtime Specification* for a complete description of DII COE developer's tools.

E.3.3 Building Your Segment

A segment must be built in a disciplined way using instructions contained in files provided with each segment. These files are contained in a special directory, `SegDescrip`, which is the segment descriptor subdirectory.

This section describes a process to turn an application into a segment so it can be a part of the DII COE. As described earlier, a segment is a collection of one or more CSCIs most conveniently managed as a unit.

E.3.3.1 Identifying and Creating Required Subdirectories

There are six segment types: Account Group, COTS, Data, Database, Software, and Patch. The following subdirectories normally are required:

Subdirectory	Description
<code>SegDescrip</code>	Subdirectory containing segment descriptor files. This directory is always required for every segment and contains the installation instructions for the segment. A segment is not allowed to directly modify any files for resources it does not <i>own</i> ; in other words, a segment cannot modify files or resources outside an assigned directory. The DII COE tools coordinate the modification of all community files at installation time.
<code>bin</code>	Executable programs for the segment. These files can be the result of a compiled program or as a result of shell scripts, depending on the type of the segment.
<code>data</code>	Subdirectory for static data items, such as menu items, that are unique to the segment, but that will be the same for all users on all workstations.

Reference Sections 5.0-5.5 of the *DII COE Integration and Runtime Specification* for a detailed explanation of segment directory layout and a description of each `SegDescrip` file.

E.3.3.2 Creating and Modifying Required Segment Descriptor Files

Segment descriptor files are the key to providing seamless and coordinated systems integration across all segments. Reference Table 1 to determine which descriptor files are required for each segment type. For example, the `AcctGrp` segment requires `ReleaseNotes`, `SegInfo`, `SegName`, and `VERSION` descriptor files in the `SegDescrip` directory, while the `Patch` segment requires the `PostInstall` descriptor file in addition to the previously listed files. Some segment descriptor information is provided in the files listed in Table 1.

NOTE: In Table 1, `Aggregate` and `COE Comp` are segment attributes that can be associated with any type of segment.

Table E-1. Segment Descriptor Files

File	Acct Grp	Aggregate	COE Comp	COTS	Data	DB	S/W Seg	Patch
DEINSTALL	O	O	O	O	O	O	O	O
FileAttribs	O	O	O	O	O	O	O	O
Installed	I	I	I	I	I	I	I	I
PostInstall	O	O	O	O	O	O	O	R
PreInstall	O	O	O	O	O	O	O	O
PreMakeInst	O	O	O	O	O	O	O	O
ReleaseNotes	R	R	R	R	R	R	R	R
SegChecksum	I	I	I	I	I	I	I	I
SegInfo	R	R	R	R	R	R	R	R
SegName	R	R	R	R	R	R	R	R
Validated	I	I	I	I	I	I	I	I
VERSION	R	R	R	R	R	R	R	R
R - Required Software O - Optional I - Created by Integrator or Installation								

Other segment descriptor information is arranged within subsections of the `SegInfo` file. As with the descriptor files themselves, some subsections of the `SegInfo` file are required and others are optional depending on the type of segment. Table 2 defines required and optional sections for each segment type.

Table E-2. SegInfo Descriptor Sections

Section	Acct Grp	Aggregate	COE Comp	COTS	Data	DB	S/W	Patch
AcctGroup	R	O	N	N	N	N	N	N
COEServices	O	O	O	O	O	O	O	O
Community	O	O	O	O	O	O	O	O
Comm.deinstall	O	O	O	O	O	O	O	O
Compat	O	O	O	O	O	O	O	N
Conflicts	O	O	O	O	O	O	O	O
Data	N	N	N	N	R	N	N	N
Database	X	X	X	X	X	X	X	X
Direct	O	O	O	O	O	O	O	O
FilesList	O	O	O	R	O	O	O	O
Hardware	R	R	R	R	R	R	R	R
Icons	R	O	N	O	N	N	O	O
Menus	R	O	N	O	N	N	O	O
ModName	*	*	*	*	*	*	*	*
ModVerify	*	*	*	*	*	*	*	*
Network	N	N	N	N	N	N	N	N
Permissions	O	O	N	N	N	N	O	O
Processes	O	O	O	O	N	N	O	O
RegrdScripts	N	N	N	N	N	N	N	N
Requires	O	O	O	O	O	O	O	O
Security	R	R	R	R	R	R	R	R
SegType	*	*	*	*	*	*	*	*
R - Required O - Optional N - Not Applicable X - Reserved for Future * - Obsolete								

E.3.3.3 Installing a Segment

Follow the procedures below to install a segment after it has been created.

Run VerifySeg

The VerifySeg tool must be run during the development phase to ensure segments use segment descriptor files properly. Run the VerifySeg tool whenever a segment is created or modified. When VerifySeg is run to verify a segment, a `Validated` file is created. Reference Appendix C, *COE Tools*, of the *DII COE Integration and Runtime Specification* for further information about using VerifySeg.

Run TestInstall

Executing the TestInstall tool is not a mandatory step in the installation process, but it is recommended. TestInstall simulates an installation of a segment on the developer's workstation before actual installation. Reference Appendix C, *COE Tools*, of the *DII COE Integration and Runtime Specification* for further information about using TestInstall.

Run TestRemove

Executing the TestRemove tool is not a mandatory step in the installation process, but it is recommended. TestRemove simulates a deinstallation of a segment on the developer's workstation after TestInstall has been run. Reference Appendix C, *COE Tools*, of the *DII COE Integration and Runtime Specification* for further information about using TestRemove.

Run MakeInstall

The MakeInstall tool is used to write one or more segments to an installation media and to package the segment(s) for distribution. MakeInstall checks if VerifySeg has been run successfully on each of the segments and aborts with an error if it has not. Reference Appendix C, *COE Tools*, of the *DII COE Integration and Runtime Specification* for further information about using MakeInstall.

Run COEInstaller

The COEInstaller tool installs a segment from floppy diskette. Reference Appendix C, *COE Tools*, of the *DII COE Integration and Runtime Specification* for further information about using the COEInstaller.

E.3.4 Customizing Your Segment

Most properly designed segments will not require any extensions to the COE, although the segments may need to add menu items and icons. Some segments may need to add special

extensions. The following subsections describe how to add menu items, icons, and special extensions.

E.3.4.1 Adding Menu Items

Menu files are maintained by the DII COE, but no DII COE applications read menu files in this release. Nevertheless, user programs may use their own menu files through this feature.

Menu Entry Format

The Menu Descriptor in the `segInfo` file is used to specify the name of the segment's menu file and the name of the affected segment's menu file.

The menu bar, pull-down menus, and cascade menus, as well as the menu items they contain, are built according to the menu description entries. The format of the entries is in ASCII with colon-separated fields. Colons are used as delimiters, and spaces are allowed in the fields. Each line ends in a colon with no extra data. A `#` symbol in the first column of a line denotes a comment line. Comment entries may be placed anywhere in the entry and are not processed by the parser.

Valid keywords are `PDMENU`, `PDMENUEND`, `ITEM`, `PRMENU`, `CASCADE`, `CASCADEEND`, `APPEND`, `APPENDEND`, and `SEPARATOR`. Since the Menu Description Entry is based on your menu design, you might not use all of these keywords. For example, if your menu does not have separator lines, your Menu Description Entry will not contain a `SEPARATOR` keyword.

Each keyword is described in the following paragraphs.

A **PDMENU** line contains the following elements:

`PDMENU: name : enable flag : id # :`

<code>PDMENU</code>	Keyword that indicates the start of a pull-down menu.
<code>name</code>	Text used to name the menu. The menu name is displayed on the menu bar.
<code>enable flag</code>	Integer value that indicates if a menu is enabled or disabled. The enable flag is 1 if a menu is enabled or 0 if it is disabled. A disabled menu means that no options under that pull-down menu can be selected.
<code>id#</code>	Optional integer value that provides a unique ID number for the menu. The <code>PDMENU id#</code> value must be unique within the menu description file. An absolute value may be provided. However, the <code>id#</code> field should be left empty so that relative numbering is used by default.

With relative numbering, an `id#` of `R1` (or leaving the field blank) sets the menu's ID number to 1 plus the `id#` of the last menu processed. An `id#` of `R2` sets the menu's ID number to 2 plus the `id#` of the last menu processed.

The following is an example of a `PDMENU` line:

```
PDMENU: Map Options : 1 : R1 :
```

A `PDMENUEND` line contains the following element:

```
PDMENUEND:
```

<code>PDMENUEND</code>	Optional keyword that indicates the end of a group of pull-down menu items. If <code>PDMENUEND</code> is not used to delimit a group of menu items, the group is presumed to end when the next keyword (other than <code>ITEM</code> or <code>PRMENU</code>) is encountered.
------------------------	---

The following is an example of a `PDMENUEND` line:

```
PDMENUEND:
```

An `ITEM` line contains the following elements:

```
ITEM: name : command : execution type : enable flag : # instances : id# :  
check value : security char : autolog flag : print flag : disk flag :
```

<code>ITEM</code>	Keyword that indicates a menu item description line.
<code>name</code>	Text used to name the menu item. The item name is displayed in the pull-down menu.
<code>command</code>	Program with space-separated arguments that is launched if the menu item type is a program. Otherwise, the menu item is called as an application callback. Because callback functions must be linked into the same executable as the menu bar, applications cannot use callbacks when adding items to the system menu bar.
<code>execution type</code>	Integer value that indicates how to execute a command, as follows: 1 = executable program 2 = void callback function with no parameters (not yet implemented) 3 = Motif callback function (not yet implemented).

<code>enable flag</code>	Integer value that indicates if a menu item is enabled or disabled. The enable flag is 1 if a menu item is enabled or 0 if it is disabled. A disabled menu item means that the option cannot be selected.
<code># instances</code>	Integer value used to set the maximum number of times the item can be executed simultaneously.
<code>id#</code>	Optional integer value that provides a unique ID number for the menu item. Each <code>ITEM id#</code> entry must be unique within a <code>PDMENU</code> listing. (<code>ITEM</code> entries in a <code>PRMENU</code> must be unique within that <code>PRMENU</code> .) Reference the <code>id#</code> description under the <code>PDMENU</code> keyword listing.
<code>check value</code>	Optional integer value that sets the star and checks annotations of a menu item. Possible values are: <ul style="list-style-type: none"> 0 = no annotation (default) 1 = visible check mark 2 = check mark, but not visible 3 = visible star 4 = star member, but not visible.
<code>security char</code>	Optional character value used to determine the lowest security level under which a menu item can be classified. Valid settings are: <ul style="list-style-type: none"> N = No Classification U = Unclassified (default) C = Confidential S = Secret T = Top Secret.
<code>autolog flag</code>	Optional character value, T or F, used to indicate if the command should be logged automatically.
<code>print flag</code>	Optional character value, T or F, used to indicate if the command should have a print capability.
<code>disk flag</code>	Optional character value, T or F, used to indicate if the command should have a disk access capability.

<p>NOTE: The following elements are not yet fully implemented: <code>check value</code>, <code>autolog flag</code>, <code>print flag</code>, and <code>disk flag</code>.</p>

The following is an example of an `ITEM` line:

```
ITEM: Netscape : Netscape.. : 1 : 1 : 1 : R1 : 0 : T : F : F : F :
```

A **PRMENU** line contains the following elements:

```
PRMENU: name : enable flag : id# :
```

<code>PRMENU</code>	Keyword that indicates a cascading menu button. It is used to mark where a cascade menu is to be connected to an upper-level menu.
<code>name</code>	Text used to name the cascade menu with which to connect. The <code>PRMENU</code> name is displayed in the pull-down menu.
<code>enable flag</code>	Integer value that indicates if a cascade menu is enabled or disabled. The enable flag is 1 if a cascade menu is enabled or 0 if it is disabled. A disabled cascade menu means that menu options on the cascade menu cannot be selected.
<code>id#</code>	Optional integer value that provides a unique ID number for the cascading menu. Each <code>PRMENU id#</code> must be unique within a <code>PRMENU</code> listing. Reference the <code>id#</code> entry under the <code>PRMENU</code> keyword listing.

The following is an example of a `PRMENU` line:

```
PRMENU: Software : 1 : R1 :
```

A **CASCADE** line contains the following element:

```
CASCADE: name :
```

<code>CASCADE</code>	Keyword that indicates the start of a cascade menu. The cascade menu connects to the <code>PRMENU</code> entry of the same name.
<code>name</code>	Text used to name a cascade menu. The name is used to attach a cascade menu to a cascading menu button. This name must be the same as the name field in the <code>PRMENU</code> entry.

The following is an example of a `CASCADE` line:

```
CASCADE: Software :
```

A **CASCADEEND** line contains the following element:

CASCADEEND :

CASCADEEND	Optional keyword that indicates the end of a group of cascade menu items. If CASCADEEND is not used to delimit a group of menu items, the group is presumed to end when the next keyword (other than ITEM or PRMENU) is encountered.
------------	--

The following is an example of a CASCADEEND line:

CASCADEEND :

An **APPEND** line contains the following elements:

APPEND: name :

APPEND	Keyword that indicates the start of a group of items to append to an existing menu. The menu will be created if it does not already exist. The group is appended to the PDMENU or CASCADE entry of the same name.
--------	---

name	Text used to select the menu to which a group of items is appended.
------	---

The following is an example of an APPEND line:

APPEND: Options :

An **APPENDEND** line contains the following element:

APPENDEND :

APPENDEND	Optional keyword that indicates the end of a group of menu items to be appended to an existing menu. If APPENDEND is not used to delimit a group of menu items, the group is presumed to end when the next keyword (other than ITEM or PRMENU) is encountered.
-----------	--

The following is an example of an APPENDEND line:

APPENDEND :

A **SEPARATOR** line contains the following element:

SEPARATOR:

SEPARATOR Optional keyword that indicates that a Motif separator widget is to be placed in a menu at the point where the keyword occurs.

The following is an example of a **SEPARATOR** line:

SEPARATOR:

Example of Adding a Menu Item

To add menu items, include the **Menus Descriptor** in the **SegInfo Segment Descriptor** file. Specify the **Menu** file you use wish to load and the **Menu** file you wish to update. The **Menu** file you wish to load should be located in the **TstSeg\data\Menu** directory, assuming the segment name is **TstSeg**. This example will add the **Test Program** menu item to the **Software** menu under the **SysAdm** account group.

The program **TSTCOEAskUser_example** will be executed when invoked through the menu item:

SegInfo (menu descriptor only)

```
[Menus]
TstSegMenu:SA_Default.main
TstSegMenu
#-----
# Software Menu Items
#-----
APPEND          :Software
ITEM            :Test Program      :TSTCOEAskUser_example:1:1:1:R1
APPENDEND :
```

Also include the **\$SEGMENT** keyword in the **SegName Segment Descriptor** file to specify the name of the affected segment. In this case it is **System Administration**

SegName

```
#
# SegName For Test Segment
#
$TYPE:SOFTWARE
$NAME:Test Segment
$PREFIX:TST
$SEGMENT:System Administration:SA:/h/AcctGrps/SysAdm
```

E.3.4.2 Adding Icons

Icon Entry Format

The Icon Description Entry contains information on all icon-based processes. The entry, or set of entries, to be used is passed to the Program Manager.

Icons are built using the icon section in the `SegInfo` file. The entry is a specially formatted icon description that has colon-separated fields. The colons are used as delimiters, and spaces are allowed in the fields. Each line ends in a colon with no extra data. A `#` symbol in the first column of a line denotes a comment line. Comment entries may be placed anywhere in the file and are not processed by the parser.

The format of the icon entry is as follows:

```
ICON file : affected icon file
```

The affected icon file contains information about both the icon and the executable. The format of the file is as follows:

```
Icon Name : Icon File : Executable File : Comments
```

Where `Icon Name` is the title placed next to the icon in the `Programs` menu, `Icon File` is the icon image file, `Executable File` is the executable to be launched by the Program Manager, and `Comments` is the comment line. The `Icon File` field is optional. If an `Icon File` is specified, the file must be located in the segment's `data\Icons` directory. The `Executable File` must be located in the segment's `bin` directory.

An example of an affected icon file is as follows:

```
Edit Profiles : EditProf.ico: EditProf.exe
```

Icons are added to a `Programs` menu group named for the account group to which they belong.

Example of Adding an Icon

To add an icon, include the `Icons Descriptor` in the `SegInfo Segment Descriptor` file. Specify the icon file you wish to load. This file should be located under the `TstSeg\data\Icons` directory, assuming the segment directory is `TstSeg`. This example will add the `Test Program` icon to the `SysAdm` account group. The program `TSTCOEAskUser_example` will be executed when invoked through the icon.

SegInfo (icon descriptor only)

```
[Icons]
TstSegIcons:SA_Default
```

TstSegIcons

```
#-----
# Software Icons
#-----
Test Program :TestProgramIcon:TSTCOEAskUser_example
```

Also include the `$SEGMENT` keyword in the `SegName` Segment Descriptor file to specify the name of the affected segment. In this case it is System Administration.

SegName

```
#
# SegName For Test Segment
#
$TYPE:SOFTWARE
$NAME:Test Segment
$PREFIX:TST
$SEGMENT:SystemAdministration:SA:/h/AcctGrps/SysAdm
```

E.3.4.3 Reserving a Socket

To add a service, include the `COEServices` Descriptor in the `SegInfo` Segment Descriptor file. Also include the `$SERVICES` keyword in the `SegInfo` Segment Descriptor file to specify the service to be added. If the port number requested is already in use under another name, an error will be generated.

NOTE: Port numbers in the range 2000-2999 are reserved for DII COE segments.

SegInfo (COEServices descriptor only)

```
[COEServices]
#
# This is my service to add
#
$SERVICES
irc_ser:3001:upd
```

E.3.4.4 Displaying a Message

This subsection shows an example of how to display a message during the PostInstall process. Five runtime tools can be used to communicate with the user: `COEAskUser`, `COEInstError`, `COEMsg`, `COEPrompt`, and `COEPromptPasswd`. These tools may be used to display information to the user or to ask the user a question and, based on the result, perform different actions.

In this example, the user is asked questions using the COEAskUser runtime tool, which is described in Appendix C, *COE Tools*, of the *DII COE Integration and Runtime Specification*.

```
rem =====
rem PostInstall script for Segment Tst
rem =====

echo "Performing PostInstall"

COEAskUser -B "RED LAN" "BLUE LAN" "Which LAN will you be connecting to?"

if ERRORLEVEL 1 goto RED

if ERRORLEVEL 0 goto BLUE

goto END:

:RED
echo "Connecting to RED LAN"
rem
rem Perform some action based user input
rem
goto END:

:BLUE
echo "Connecting to BLUE LAN"
rem
rem Perform some action based on user input
rem

:END
echo "Done with PostInstall"
```

Attachment A - Sample Segments

This attachment includes a segment layout for a sample Account Group segment (GCCS) and a sample Software segment (Seg1). These basic templates of typical DII COE segments can be used to test segment installation and execution.

NOTE: If you do not have the GCCS sample Account Group segment installed on your machine, you will receive several warnings indicating that it must be installed before the Seg1 sample Software segment can be loaded.

The Seg1 sample Software segment will add the Seg1 Hello World icon to the GCCS Account Group in the Programs menu. The program Seg1_HelloWorld.exe will be executed when invoked through the icon.

Reference Attachment B, *Verifying Segment Syntax and Loading a Segment onto a Floppy Diskette*, for instructions on how to validate the Seg1 sample Software segment and load the segment onto a floppy diskette.

AA.1 Sample Account Group Segment Layout

The layout of the GCCS sample Account Group segment is:

```
gccs

  SegDescrip
    DEINSTALL.BAT
    PostInstall.bat
    ReleaseNotes
    SegInfo
    SegName
    VERSION
```

The SegDescrip files contain the following:

```
DEINSTALL.BAT
REM =====
REM
REM DEINSTALL
REM
REM Routine to perform necessary actions when segment
REM is deinstalled.
REM
REM =====
```

PostInstall.bat

```
REM =====
REM
REM PostInstall
REM
REM Routine to perform necessary actions after segment
REM has been loaded.
REM
REM =====
```

ReleaseNotes

This is a sample Account Group.
Other Segments will need to extend the environment
as they add their specific functionality
to the account group.

SegInfo

```
#=====
#
#   Account Group SegInfo file.
#
#=====
```

```
[AcctGroup]
GCCS Operator:350::1:GCCS:GCCS Default
$CLASSIF:UNCLASS
```

```
[Hardware]
$CPU:PC
$OPSYS:NT
$DISK:500
$MEMORY:100
```

```
[Security]
UNCLASS
```

SegName

```
#=====
#
#   Account Group SegName file.
#
#=====
$TYPE:ACCOUNT GROUP
$NAME:GCCS COE
$PREFIX:GCCS
```

VERSION

3.0.0.6:2/5/97

AA.2 Sample Software Segment Layout

The layout of the Seg1 sample Software segment is:

```
Seg1
  bin
    Seg1_HelloWorld.exe
  data
    Icons
      TestIcons
  SegDescrip
    DEINSTALL.BAT
    PostInstall.BAT
    ReleaseNotes
    SegInfo
    SegName
    VERSION
```

The SegDescrip files contain the following:

```
DEINSTALL
REM =====
REM
REM DEINSTALL
REM
REM Routine to perform necessary actions when Seg1
REM is deinstalled.
REM
REM =====
```

```
PostInstall.BAT
REM =====
REM
REM PostInstall
REM
REM Routine to perform necessary actions after Seg1 Test
REM Segment has been loaded.
REM
REM =====
```

```
ReleaseNotes
This is the Seg1 Test Segment.
```

SegInfo

```
#=====
#
#   DII Database Admin Segment SegInfo
#   Descriptor file.
#
#=====
[Hardware]
$CPU:PC
$OPSYS:NT
$DISK:500
$MEMORY:100

[Icons]
TestIcons

[Security]
UNCLASS
```

SegName

```
#=====
#
#   DII Seg1 Test Segment
#   Descriptor file.
#
#=====
$TYPE:SOFTWARE
$NAME:Test Segment #1
$PREFIX:Seg1
$SEGMENT:GCCS COE:GCCS:/h/AcctGrps/GCCS
```

VERSION

3.0.0.6:10/15/96

The data\Icons file contains the following:

TestSegIcons

```
TstSegIcons
#-----
# Software Icons
#-----
Test Program :TestProgramIcon.ico:TSTCOEAskUser_example
```

Attachment B - Verifying Segment Syntax and Loading a Segment onto a Floppy Diskette

This attachment provides examples of how to verify segment syntax, install a segment temporarily, and load a segment onto an installation floppy diskette. The segment verification and loading process involves the following steps:

- STEP 1: **Run the VerifySeg tool.** Run VerifySeg to validate that the segment conforms to the rules for defining a segment (i.e., to verify the segment syntax).
- STEP 2: **Run the TestInstall tool.** Run TestInstall against the sample segment to install the segment temporarily. This step is optional; if you choose not to run TestInstall, proceed to STEP 4.
- STEP 3: **Run the TestRemove tool.** Run TestRemove against the sample segment to remove the segment temporarily installed in STEP 2.
- STEP 4: **Run the MakeInstall tool.** Run MakeInstall to load the segment onto floppy diskette. After the segment is loaded onto floppy diskette, it is ready to be installed using the `DII Installer` icon from the System Administration group.

Subsections AB.1-AB.4 show how to perform these steps against the Seg1 sample Software segment, which is described in Attachment A, *Sample Segments*.

NOTE: In the following subsections, the VerifySeg, TestInstall, TestRemove, and MakeInstall tools are being run against the Seg1 sample Software segment. The output of each command will vary depending on the segment being converted. Note the following severity indicators:

- | | |
|-----------------------------|-----------------------------------|
| (F) indicates a FATAL ERROR | (D) indicates a DEBUG statement |
| (W) indicates a WARNING | (V) indicates a VERBOSE statement |
| (E) indicates an ERROR | (O) indicates an ECHO statement. |

NOTE: In the following subsections, boldface text indicates information that the user must input.

AB.1 Running VerifySeg Against the Sample Segment

```
*****
VerifySeg -p \SampleSegs Seg1

Results of verification (/SampleSegs/Seg1) :
  Totals
  -----
  Errors:      0
  Warnings:    0
*****
```

AB.2 Running TestInstall Against the Sample Segment

```
*****
TestInstall -p \SampleSegs Seg1
*****
TestInstall - Version 1.0.0.7
*****
The following options have been selected:
*****
Print warning messages.
*****
Segments to be TestInstalled:
*****
Segment: seg1 Path: P:\SampleSegs
*****WARNING*****
TestInstall may modify COE files already in use.
This may cause unpredictable results if COE processes are already running.
Make sure no other COE processes are running before using TestInstall.
Do you want to continue with the TestInstall? (y/n):y
Processing seg1

  Successfully ran preprocessor on segment seg1
No PreInstall script for segment seg1
Do you want to run PostInstall for Segment seg1? (y/n):y
REM =====
REM
REM PostInstall
REM
REM Routine to perform necessary actions after Seg1 Test
REM Segment has been loaded.
REM
REM =====
Successful Installation of seg1
*****
```

AB.3 Running TestRemove Against the Sample Segment

```
*****
TestRemove -p \SampleSegs Seg1
*****WARNING*****
TestRemove may modify COE files already in use.
This may cause unpredictable results if COE processes are already running.
Make sure there are no other COE processes running before using TestRemove.
Do you want to continue with the TestRemove? (y/n):y
SETTING P:\SampleSegs\Seg1 FOR INSTALL_DIR

REM =====
REM
REM  DEINSTALL
REM
REM  Routine to perform necessary actions when Seg1
REM  is deinstalled.
REM
REM =====
*****
Successful Removal of Seg1
```

AB.4 Running MakeInstall Against the Sample Segment

The example below shows the three windows displayed by MakeInstall as it loads segments onto a floppy diskette. The MakeInstall tool does not generate any command window output. The Seg1 sample Software segment is used in the example. These windows should appear in the order shown below.

```
*****
MakeInstall -p d:\tmp Seg1
```

Make Install

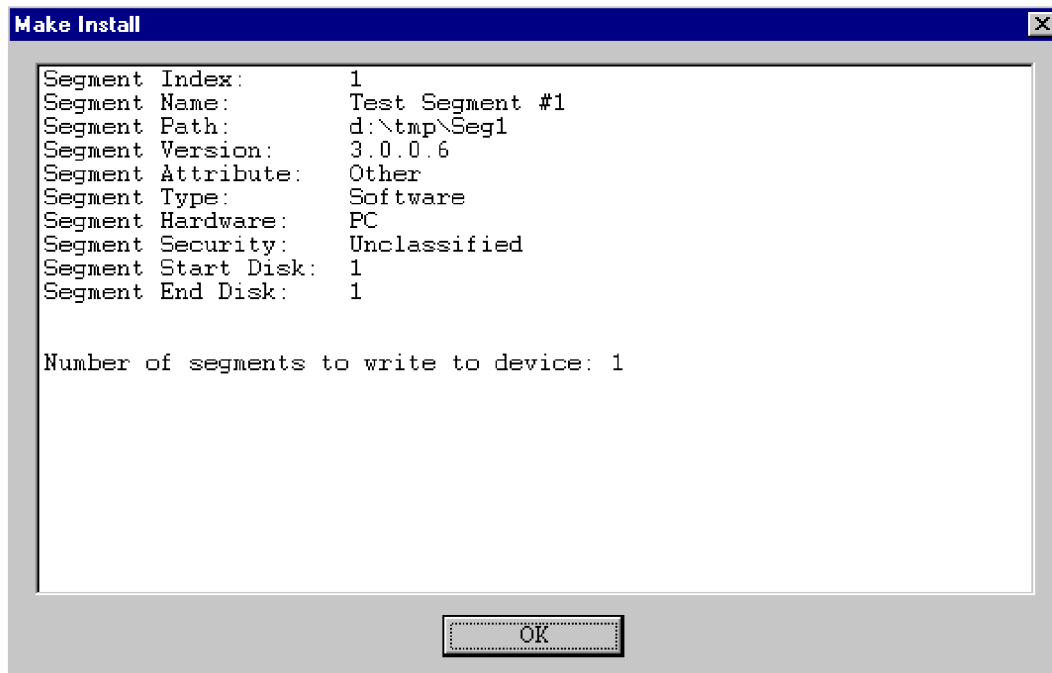
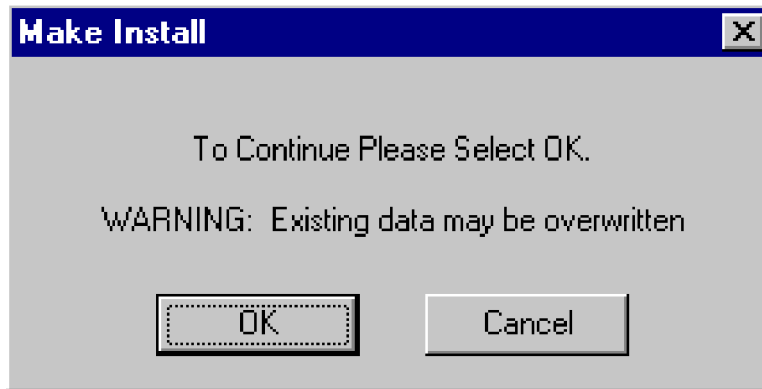
Please enter the following information for the Tape Header:

Name: Developer

Serial: 1

Comment: Test Load

OK Cancel



Attachment C - Installing the Developer's Toolkit

The Developer's Toolkit contains the components needed to create segments that use DII COE components. Developer's tools are delivered on floppy diskettes separate from the DII COE Kernel. These tools can be run from the command line.

Follow the steps below to install and run the DII COE 3.0.0.6 Developer's Toolkit for Windows NT 4.0:

STEP 1: Choose a directory for installation. Choose a directory on your hard drive into which the tools can be installed.

NOTE: These developer's tools are not location sensitive and can be moved to any directory desired for development; however, the provided Dynamic Link Libraries (DLLs) must remain in the same directory with the tools that require them or must be in a location that is also included in the `PATH` environment variable. Therefore, it is recommended that the tools remain in the same directory (i.e., `DII_DEV\bin`).

STEP 2: Copy the `DII_DEV` directory to the desired location. Use the Windows NT Explorer to copy the contents of the `DII_DEV` directory from the Developers' Toolkit floppy diskettes to the desired location.

NOTE: To copy a directory, click on the `Start` button on the task bar, point to `Programs`, and click on the `Windows NT Explorer` option. The `Exploring` window appears. Click on the directory you want to copy to highlight it and then select the `Copy` option from the `Edit` pull-down menu. Open the destination directory and select `Paste` from the `Edit` pull-down menu. The directory, then, is copied to the new location.

STEP 3: Add an entry in your `PATH` environment variable. Add an entry in your `PATH` environment variable for the `DII_DEV\bin` directory. To add this entry, click on the `Start` button on the task bar, point to `Settings`, and click on `Control Panel`. The `Control Panel` window appears. Double-click on the `System` icon to open the `System Properties` window. Click on the `Environment` tab.

If the `PATH` environment variable appears in the `User Variables` for `Administrator` panel, highlight it. The word `Path` should appear in the `Variable` field and a list of one or more paths separated by semicolons should appear in the `Value` field. Place your cursor at the end of this list of paths and type `:[directory]\DII_DEV\bin` in the `Value` field, where `[directory]` is the directory on your hard drive in which you installed the developer's toolkit.

If the `PATH` environment variable does not appear in the `User Variables` for `Administrator` panel, type `Path` in the `Variable` field and type `[directory]\DII_DEV\bin` in the `Value` field.

Click on the **Set** button to establish the path as a setting in the **User Variables for Administrator** panel. Then click on the **OK** button to exit the window.

STEP 4: Create or copy a DII COE segment onto your hard drive. Reference the *DII COE Integration and Runtime Specification* for information about creating a segment in the DII COE format.

STEP 5: Run VerifySeg. Type the following command to run VerifySeg:

```
VerifySeg -p [segment path] [segment directory]
```

STEP 6: Run TestInstall. Type the following command to run TestInstall:

```
TestInstall -p [segment path] [segment directory]
```

STEP 7: Run TestRemove. Type the following command to run TestRemove:

```
TestRemove -p [segment path] [segment directory]
```

STEP 8: Run MakeInstall. Type the following command to run MakeInstall:

```
MakeInstall -p [segment path] [segment directory]
```